aws

Panduan Developer

AWS Lambda



Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

AWS Lambda: Panduan Developer

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Merek dagang dan tampilan dagang Amazon tidak boleh digunakan sehubungan dengan produk atau layanan apa pun yang bukan milik Amazon, dengan cara apa pun yang dapat menyebabkan kebingungan di antara pelanggan, atau dengan cara apa pun yang menghina atau mendiskreditkan Amazon. Semua merek dagang lain yang tidak dimiliki oleh Amazon adalah milik dari pemiliknya masing-masing, yang mungkin berafiliasi atau tidak berafiliasi dengan, terkait, atau disponsori oleh Amazon.

Table of Contents

Apa itu AWS Lambda? 1
Bagaimana Lambda menghitung primitif dibandingkan 2
Panduan penyiapan agen 5
Prasyarat 5
Kode Claude 5
Instal plugin aws-core dari Agen Toolkit untuk AWS 5
kodeks 6
Instal plugin aws-core dari Agen Toolkit untuk AWS 6
Kursor 6
Menginstal AWS keterampilan tanpa server 6
Kiro 6
Menginstal AWS keterampilan tanpa server di Kiro CLI 6
Instal kekuatan Kiro (dari Kiro IDE) 6
GitHub Kopilot 7
Menginstal AWS keterampilan tanpa server 7
Devin Desktop 7
Menginstal AWS keterampilan tanpa server 7
OpenCode 7
Menginstal AWS keterampilan tanpa server 7
Untuk semua agen lain yang kompatibel dengan keterampilan agen 7
Menginstal AWS keterampilan tanpa server 7
Instal keterampilan Lambda tambahan 8
Fungsi Acara 10
Kapan menggunakan Fungsi Acara 10
Fungsi Acara dan fungsi tahan lama 11
Bagaimana Lambda bekerja 11
Fitur utama 12
Informasi Terkait 13
Konsep inti 13
Fungsi lambda dan penangan fungsi 14
Lingkungan eksekusi Lambda dan runtime 15
Peristiwa dan pemicu 15
Izin dan peran Lambda 16
Menjalankan kode 19

Membuat arsitektur berbasis peristiwa 34
Merancang aplikasi 47
Buat fungsi pertama Anda 55
Mendaftar untuk Akun AWS 55
Buat fungsi 55
Invokasi fungsi 62
Bersihkan 65
Langkah selanjutnya 66
Contoh aplikasi dan pola 67
Pemrosesan File 67
Integrasi Database 68
Tugas Terjadwal 68
Sumber daya tambahan 68
File-processing aplikasi 69
Scheduled-maintenance aplikasi 93
Bahasa pemrograman 113
Membangun dengan Node.js 114
Membangun dengan TypeScript 191
Membangun dengan Python 241
Membangun dengan Ruby 341
Membangun dengan Java 408
Membangun dengan Go 522
Membangun dengan C # 582
Membangun dengan PowerShell 671
Membangun dengan Karat 686
Waktu berjalan 711
Waktu aktif yang didukung 712
Rilis runtime baru 716
Kebijakan penghentian waktu proses 716
Model tanggung jawab bersama 717
Penggunaan runtime setelah penghentian 719
Menerima pemberitahuan penghentian runtime 721
Waktu pengoperasian terdepresiasi 722
Pembaruan versi runtime 726
Dapatkan data tentang fungsi berdasarkan runtime 738
Modifikasi waktu pengoperasian 745

API runtime 749
OS-only runtime 758
Repositori sumber terbuka 773
Mengkonfigurasi Fungsi Acara 776
arsip file .zip 779
Image kontainer 792
Self-managed Penyimpanan kode S3 802
Memori 808
Penyimpanan sementara 811
Set instruksi (ARM/x86) 814
Batas waktu 818
Variabel lingkungan 821
Melampirkan fungsi ke VPC 835
Melampirkan fungsi ke sumber daya di akun lain 851
Akses internet untuk fungsi VPC 858
Jaringan masuk 883
Sistem file 888
Alias 897
Versi 905
Tanda 910
Streaming respons 914
Titik akhir metadata 925
Memanggil Fungsi Acara 930
Invokasi sinkron 932
Invokasi asinkron 936
Pemetaan sumber peristiwa 950
Pemfilteran peristiwa 962
Pengujian di konsol 978
Status fungsi 982
Percobaan ulang 985
Deteksi loop rekursif 987
URL fungsi 997
Penskalaan Fungsi Acara 1049
Memahami dan memvisualisasikan konkurensi 1049
Menghitung konkurensi untuk Fungsi Acara 1054
Memahami konkurensi cadangan dan konkurensi yang disediakan 1056

Memahami konkurensi dan permintaan per detik 1065
Kuota konkurensi 1067
Mengonfigurasi konkurensi terpesan 1070
Mengonfigurasi konkurensi yang tersedia 1075
Perilaku penskalaan 1086
Memantau konkurensi 1088
Pemantauan dan debugging 1094
Harga 1094
Metrik Fungsi Acara 1095
Log Fungsi Acara 1107
CloudTrail log 1147
AWS X-Ray 1156
Wawasan Fungsi Acara 1165
Lihat metrik aplikasi 1174
Sinyal Aplikasi 1176
Debug dengan VS Code 1180
Mengintegrasikan layanan lain 1183
Membuat pemicu 1184
Daftar layanan 1184
Apache Kafka 1187
API Gateway 1353
Komposer Infrastruktur 1384
CloudFormation 1387
Amazon DocumentDB 1390
DynamoDB 1429
EC2 1492
Elastic Load Balancing (Application Load Balancer) 1494
Memanggil menggunakan Scheduler EventBridge 1497
IoT 1500
Kinesis Data Streams 1502
Kubernetes 1564
MQ 1566
RDS 1588
S3 1620
Secrets Manager 1676
SQS 1695

Step Functions 1757
S3 Batch 1767
SNS 1770
Fungsi tahan lama 1792
Manfaat utama 1793
Kapan menggunakan fungsi tahan lama 1793
Cara Kerjanya 1794
Langkah selanjutnya 1796
Konsep dasar 1797
Menciptakan fungsi yang tahan lama 1801
Konfigurasikan fungsi yang tahan lama 1829
Fungsi tahan lama atau Step Functions 1832
Contoh 1835
Keamanan dan izin 1860
Mengenkripsi data eksekusi yang tahan lama 1867
SDK eksekusi tahan lama 1883
Memohon fungsi yang tahan lama 1893
Menguji fungsi tahan lama 1917
Memantau fungsi tahan lama 1918
Praktik terbaik 1924
SnapStart 1926
Kasus penggunaan 1927
Fitur yang didukung dan batasan 1928
Wilayah yang Didukung 1928
Pertimbangan kompatibilitas 1928
Harga 1929
Mengaktifkan SnapStart 1931
Menangani keunikan 1938
Kait runtime 1945
Memantau 1958
Model keamanan 1961
Praktik terbaik 1962
Pemecahan masalah 1968
Isolasi penyewa 1971
Kapan menggunakan mode isolasi penyewa 1972
Fitur yang didukung dan batasan 1972

Didukung Wilayah AWS 1973
Pertimbangan-pertimbangan 1973
Harga 1973
Mode isolasi 1973
Mengaktifkan isolasi penyewa 1975
Memanggil fungsi dengan isolasi penyewa 1978
Mengakses dalam kode 1982
Memantau 1986
Pemecahan masalah 1988
Instans Terkelola 1988
Kemampuan Utama 1989
Kapan menggunakan Instans Terkelola Lambda 1989
Cara Kerjanya 1990
Model konkurensi 1990
Sewa dan isolasi 1990
Harga 1991
Bagaimana Instans Terkelola Lambda berbeda dari tipe komputasi Lambda (default) 1991
Langkah selanjutnya 1993
Memulai 1993
Konsep inti 2000
Deployment 2026
Waktu berjalan 2035
Jaringan 2060
Memantau 2064
Kuota 2080
Praktik terbaik 2083
Pemecahan masalah 2087
Lapisan 2097
Cara menggunakan lapisan 2099
Versi lapisan dan lapisan 2099
Lapisan kemasan 2100
Membuat dan menghapus lapisan 2104
Menambahkan lapisan 2107
Lapisan dengan CloudFormation 2111
Lapisan dengan AWS SAM 2112
Ekstensi 2112

Lingkungan eksekusi 2113
Dampak pada performa dan sumber daya 2115
Izin 2115
Mengonfigurasi ekstensi 2116
Mitra ekstensi 2118
API Ekstensi 2120
API Telemetri 2138
Pemecahan masalah 2194
Konfigurasi 2195
Deployment 2204
Invokasi 2212
Eksekusi 2220
Pemetaan sumber peristiwa 2230
Jaringan 2236
Praktik terbaik 2238
Kode Fungsi Acara 2239
Konfigurasi Fungsi Acara 2241
Skalabilitas Fungsi Acara 2242
Metrik dan alarm 2242
Bekerja dengan stream 2243
Praktik terbaik keamanan 2245
Aplikasi sampel 2245
Alat pengembangan 2248
Alat pengembangan lokal 2249
Infrastruktur sebagai alat Kode (IAc) 2250
GitHub Alat tindakan 2250
Powertools untuk AWS Lambda 2251
Alur kerja dan alat manajemen acara 2251
Pembangunan lokal 2251
GitHub Tindakan 2260
Infrastruktur sebagai kode (IAc) 2261
Powertools untuk AWS Lambda 2285
Alur kerja dan acara 2286
Pengujian 2289
Hasil bisnis yang ditargetkan 2290
Apa yang harus diuji 2291

Cara menguji tanpa server ... 2292
Teknik pengujian ... 2292
Praktik terbaik ... 2299
Tantangan pengujian secara lokal ... 2303
Pertanyaan yang Sering Diajukan ... 2304
Langkah dan sumber daya selanjutnya ... 2305
Izin ... 2306
Peran eksekusi (izin untuk fungsi untuk mengakses sumber daya lain) ... 2309
Izin akses (izin untuk entitas lain untuk mengakses fungsi Anda) ... 2322
Fungsi Web ... 2367
Kapan menggunakan Fungsi Web ... 2368
Cara Kerja Fungsi Web ... 2369
Manfaat utama ... 2369
Konsep inti ... 2370
Model sumber daya ... 2370
Status revisi ... 2372
Status titik akhir ... 2373
Lingkungan eksekusi ... 2374
Permintaan penanganan ... 2376
Revisi ... 2376
Titik akhir ... 2387
Buat Fungsi Web pertama Anda ... 2393
Prasyarat ... 2394
Membuat Fungsi Web (konsol) ... 2394
Membuat Fungsi Web (AWS CLI) ... 2396
Bersihkan ... 2401
Deployment ... 2402
Kode di Amazon S3 ... 2402
Pola deployment ... 2408
Multi-Region penyebaran ... 2416
Perilaku HTTP ... 2420
Metode HTTP ... 2420
Minta header ... 2421
Header respons ... 2422
Batas header ... 2422
Jaringan ... 2423

Konektivitas Masuk 2423
Konektivitas keluar 2426
Menghubungkan Fungsi Web ke VPC Anda 2427
Memperbarui atau menghapus konektivitas VPC 2431
Dukungan IPv6 2432
Penskalaan 2432
Lingkungan eksekusi 2433
Batas konkurensi dinamis per lingkungan 2433
Pengaturan penskalaan 2434
Kuota akun 2436
Bagaimana pengaturan penskalaan berinteraksi 2436
Throttling 2438
Menaikkan dan menurunkan skala 2439
Memilih pengaturan penskalaan 2439
Penskalaan untuk MultiRegion dan PerRegion titik akhir 2441
Contoh penskalaan 2441
Siklus hidup lingkungan 2446
Fase siklus hidup 2446
Inisialisasi (start dingin) 2447
Scale-down dan shutdown 2448
Penggunaan kembali lingkungan 2449
Penanganan permintaan bersamaan 2449
Runtime 2450
Didukung Node.js versi 2450
Penambalan runtime otomatis 2450
Kerangka kerja web yang didukung 2451
Node.js model eksekusi 2451
Memori dan Node.js tumpukan 2452
Bagaimana Lambda merutekan permintaan ke aplikasi Anda 2453
Mengkonfigurasi Fungsi Web 2453
Paket deployment 2454
Arsitektur 2455
Batas waktu 2455
Variabel lingkungan 2455
Tanda 2458
Titik akhir metadata 2458

Izin 2460
Peran eksekusi 2461
Resource-based kebijakan 2464
Izin untuk mengkonfigurasi jaringan 2471
Kontrol akses 2472
Menggunakan jenis `IamAuth`otentikasi 2472
Menggunakan jenis `ApplicationManaged`otentikasi 2477
Tata kelola dan kontrol akses 2478
Enkripsi saat diam 2480
Opsi untuk enkripsi saat istirahat 2480
Mengenkripsi data saat istirahat menggunakan kunci yang dikelola pelanggan 2481
Memantau interaksi Fungsi Web dengan AWS KMS 2491
Memantau 2492
Penagihan telemetri 2493
Pertimbangan telemetri 2494
Metrik-metrik 2496
Beberapa catatan 2503
Metrik-metrik kustom 2522
CloudTrail log 2525
Aplikasi sampel 2535
Minimal Ekspres application 2535
Hono API dan file statis 2536
Server-sent acara 2537
Multi-Region rute verifikasi 2537
Package dan menyebarkan sampel 2537
Praktik terbaik 2538
Amankan setiap titik akhir 2538
Pertahankan artefak penyebaran yang tidak dapat diubah 2539
Desain untuk permintaan bersamaan 2539
Jaga agar startup dapat diprediksi 2540
Tetapkan batas dari perilaku terukur 2540
Terapkan dengan verifikasi eksplisit dan rollback 2540
Membangun baseline operasional 2541
Beroperasi MultiRegion dan PerRegion titik akhir sebagai sistem terdistribusi 2541
Pengujian 2542
Lapisan uji yang direkomendasikan 2542

Jalankan server HTTP secara lokal 2543
Uji paket penerapan 2543
Uji di lingkungan cloud yang terisolasi 2544
Uji otorisasi dan eksposur 2544
Uji konkurensi, penskalaan, dan pelambatan 2544
Uji penerapan dan rollback 2545
Pemecahan masalah 2545
Kumpulkan informasi diagnostik 2546
Masalah penyebaran dan startup 2547
Kesalahan CLI dan paket 2548
Kegagalan permintaan HTTP 2549
Minta batas header 2553
Jenis kesalahan Fungsi Web 2555
Log atau metrik tidak ada 2558
Masalah penyimpanan dan enkripsi kode 2559
Masalah dengan `MultiRegion`dan titik `PerRegion`akhir 2559
Masalah jaringan dan konektivitas 2560
Siapkan bundel eskalasi 2561
Informasi Terkait 2562
MicroVM 2563
Cara kerja Lambda MicroVMS 2563
Kapan menggunakan Lambda MicroVMS 2564
Fitur utama 2564
Konsep inti 2565
Konsep Utama 2565
Bagaimana Lambda membangun citra Anda 2566
Siklus hidup MicroVM 2567
Buat MicroVM pertama Anda 2569
Prasyarat 2569
Membuat Image MicroVM Pertama Anda 2570
Menjalankan MicroVM pertama Anda 2573
Menghubungkan ke MicroVM Anda 2574
Bersihkan 2574
Langkah selanjutnya 2574
Image MicroVM 2575
Bagaimana Lambda membangun image microVM 2575

Ukuran MicroVM 2576
Gambar dasar MicroVM 2576
Kait pembuatan gambar MicroVM 2578
Memperbarui gambar microVM 2579
Status gambar dan membangun status 2580
Variabel lingkungan 2581
Snapshot 2581
Penambalan gambar MicroVM 2583
Gambar basis kontainer 2583
Kemampuan sistem operasi 2584
Menjalankan MicroVMS 2585
Memulai MicroVM 2585
Menghubungkan ke MicroVM 2588
Pengait siklus hidup 2591
Menangguhkan dan Melanjutkan MicroVM 2594
Penskalaan dan konkurensi 2596
Mengakhiri MicroVM 2596
Daftar MicroVM 2597
Penanganan kesalahan 2597
Jaringan 2598
Gambaran umum 2598
Konektivitas Masuk 2599
Konektivitas keluar 2602
Bekerja dengan konektor jaringan jalan keluar 2602
Menggunakan MicroVMS Lambda dengan titik akhir VPC antarmuka (AWS PrivateLink) ... 2606
Keamanan 2611
Peran IAM 2611
Token otentikasi 2612
Token akses shell 2612
Menandai sumber daya 2613
Referensi izin IAM 2613
Format ARN sumber daya 2614
Contoh: Least-privilege kebijakan untuk operator 2614
Memantau 2615
Pencatatan (CloudWatch Log) 2615
CloudTrail penebangan 2616

Pemecahan masalah 2620
Akses shell 2620
Pemecahan masalah 2621
Kesalahan umum (pembuatan gambar) 2622
Pemecahan masalah konektor jaringan 2623
Praktik terbaik 2624
Desain aplikasi 2625
Performa 2625
Keamanan 2625
Optimalisasi biaya 2626
Memantau 2626
Integrasi 2626
Agen Terkelola Claude 2627
Agen Cloud Cursor 2631
API Agen OpenAI 2637
Keamanan, tata kelola, dan kepatuhan 2647
Perlindungan data 2648
Enkripsi dalam transit 2649
Enkripsi saat diam 2650
Menggunakan peran tertaut layanan 2655
Service-linked izin peran untuk Lambda 2656
Membuat peran terkait layanan untuk Lambda 2656
Mengedit peran terkait layanan untuk Lambda 2657
Menghapus peran terkait layanan untuk Lambda 2657
Wilayah yang Didukung untuk peran terkait layanan Lambda 2658
Identity and Access Management 2659
Audiens 2660
Mengautentikasi dengan identitas 2660
Mengelola akses menggunakan kebijakan 2662
Bagaimana AWS Lambda bekerja dengan IAM 2663
Identity-based contoh kebijakan 2669
AWS kebijakan terkelola 2672
Pemecahan masalah 2684
Tata kelola 2686
Kontrol proaktif dengan Guard 2689
Kontrol proaktif dengan AWS Config 2693

Kontrol detektif dengan AWS Config 2701
Penandatanganan kode 2706
Pemindaian kode 2709
Observabilitas 2714
Validasi kepatuhan 2723
Ketahanan 2723
Keamanan infrastruktur 2724
Mengamankan beban kerja dengan titik akhir publik 2725
Autentikasi dan otorisasi 2725
Melindungi titik akhir API 2726
Penandatanganan kode 2727
Validasi tanda tangan 2728
Membuat Konfigurasi 2729
Izin 2731
Tag konfigurasi penandatanganan kode 2732
Bekerja dengan AWS SDK 2737
Contoh kode menggunakan SDK 2739
Hal-hal mendasar 2742
Halo Lambda 2743
Pelajari dasar-dasarnya 2752
Tindakan 2889
Skenario 3023
Secara otomatis mengonfirmasi pengguna yang dikenal dengan fungsi Lambda 3024
Secara otomatis memigrasikan pengguna yang dikenal dengan fungsi Lambda 3065
Buat REST API untuk melacak COVID-19 data 3088
Membuat API REST pustaka peminjaman 3089
Membuat aplikasi messenger 3090
Buat API rest dengan integrasi proxy fungsi 3091
Membuat aplikasi nirserver untuk mengelola foto 3099
Membuat aplikasi obrolan websocket 3103
Buat aplikasi untuk menganalisis umpan balik pelanggan 3104
Membuat dasbor pemantauan dengan nama fungsi sebagai variabel 3110
Membuat fungsi tanpa server pertama Anda 3121
Menginvokasi fungsi Lambda dari browser 3133
Transformasi data dengan S3 Object Lambda 3134
Menggunakan API Gateway untuk menginvokasi fungsi Lambda 3134

Menggunakan Step Functions untuk menginvokasi fungsi Lambda 3136
Menggunakan peristiwa terjadwal untuk menginvokasi fungsi Lambda 3137
Gunakan API Neptunus untuk menanyakan data grafik 3139
Menggunakan variabel properti dalam dasbor pemantauan untuk memantau beberapa fungsi tanpa server 3140
Tulis data aktivitas khusus dengan fungsi Lambda setelah otentikasi pengguna Amazon Cognito 3149
Contoh nirserver 3171
Menghubungkan ke database Amazon RDS dalam fungsi Lambda 3172
Memanggil fungsi Lambda dari pemicu Kinesis 3191
Memanggil fungsi Lambda dari pemicu DynamoDB 3201
Memanggil fungsi Lambda dari pemicu Amazon DocumentDB 3211
Memanggil fungsi Lambda dari pemicu Amazon MSK 3223
Menginvokasi fungsi Lambda dari pemicu Amazon S3 3233
Memanggil fungsi Lambda dari pemicu Amazon SNS 3245
Memanggil fungsi Lambda dari pemicu Amazon SQS 3254
Melaporkan kegagalan item batch untuk fungsi Lambda dengan pemicu Kinesis 3263
Melaporkan kegagalan item batch untuk fungsi Lambda dengan pemicu DynamoDB 3276
Melaporkan kegagalan item batch untuk fungsi Lambda dengan pemicu Amazon SQS 3287
AWS kontribusi masyarakat 3297
Membangun dan menguji aplikasi tanpa server 3297
Kuota Lambda 3300
Komputasi dan penyimpanan 3302
Konfigurasi fungsi, penerapan, dan eksekusi 3303
Permintaan API Lambda 3306
Fungsi tahan lama 3308
MicroVM 3310
Komputasi dan penyimpanan 3310
Gambar dan versi 3311
Per-MicroVM throughput 3312
Batas tarif API 3312
Fungsi Web Lambda 3313
Fungsi Web, revisi, dan kuota lingkungan eksekusi 3313
Kuota titik akhir 3315
Kuota akun 3316
Kendala jaringan 3316

Layanan lainnya 3317
Riwayat dokumen 3318
Pembaruan sebelumnya 3349
..... mmmccclvi

Apa itu AWS Lambda?

AWS Lambda adalah layanan komputasi tanpa server. Dengan Lambda, Anda dapat menjalankan kode tanpa menyediakan atau mengelola server. Lambda secara otomatis mengelola infrastruktur yang mendasarinya — termasuk pemeliharaan server, penyediaan kapasitas, penskalaan, dan penambalan — sehingga Anda dapat fokus pada logika aplikasi Anda.

Lambda menyediakan tiga primitif komputasi, masing-masing dirancang untuk pola beban kerja yang berbeda:

- Fungsi Peristiwa Lambda - Jalankan kode sebagai respons terhadap peristiwa atau panggilan API tanpa mengelola server. Anda menulis fungsi handler, menghubungkannya ke pemicu (API Gateway, Amazon S3, Amazon SQS, dan 200+ AWS layanan lainnya) EventBridge, dan Lambda menjalankannya. Setiap pemanggilan berjalan secara independen tanpa status bersama, menskalakan secara horizontal agar sesuai dengan permintaan. Lambda mengelola lingkungan eksekusi, penskalaan, perutean, dan toleransi kesalahan.
- Fungsi Web Lambda — Jalankan aplikasi web dan backend API sebagai server HTTP. Node.jsKode Anda mendengarkan pada port dan menangani permintaan HTTP secara langsung, menggunakan kerangka kerja web sepertiExpress,Hono, atau React Router tanpa adaptor. Setiap lingkungan eksekusi melayani beberapa permintaan bersamaan. Lambda menyediakan titik akhir HTTPS, menskalakan aplikasi Anda berdasarkan lalu lintas (termasuk ke nol), dan dapat mereplikasi semuanya. Wilayah AWS
- Lambda MicroVMS — Lingkungan komputasi terisolasi dengan startup hampir instan dan umur total hingga 8 jam (termasuk waktu berjalan dan ditangguhkan). Dirancang untuk beban kerja yang membutuhkan lingkungan komputasi khusus untuk setiap pengguna atau pekerjaan individu. Lambda mengelola isolasi, kapasitas, dan jaringan. Aplikasi Anda menggunakan API MicroVMS Lambda dan titik akhir HTTPS untuk menghubungkan masing-masing user/job ke lingkungan komputasi mereka.

> **Note**
>
> Fungsi Acara adalah nama untuk jenis fungsi Lambda asli yang menjalankan kode Anda sebagai respons terhadap peristiwa dan panggilan API. Nama ini membedakan jenis fungsi ini dari Lambda Web Functions dan Lambda MicroVMS, dan konsol Lambda memberi label fungsi Event. Sumber daya yang mendasarinya tidak berubah: operasi API (seperti `CreateFunction` dan`Invoke`), AWS SDK, AWS CLI, tipe `AWS::Lambda::Function`

sumber daya, dan ARN terus menggunakan fungsi, jadi Anda tidak perlu mengubah kode, templat infrastruktur, atau izin IAM apa pun.

Untuk informasi harga, lihat Harga AWS Lambda.

Bagaimana Lambda menghitung primitif dibandingkan

Fungsi Acara Lambda, Fungsi Web Lambda, dan MicroVMS Lambda berbagi fondasi tanpa server yang sama:

- Tidak ada manajemen server — AWS mengelola infrastruktur yang mendasarinya, penambalan instance, dan kapasitas.
- Pay-per-use penagihan - Tidak ada komitmen di muka. Anda hanya membayar untuk sumber daya yang digunakan.
- Jaringan terkelola - Ketiganya menyediakan akses jaringan masuk dan keluar yang dikelola layanan.
- Virtualisasi petasan — VM-level isolasi antara beban kerja.

Sementara mereka berbagi yayasan ini, mereka melayani kasus penggunaan yang berbeda:

	Fungsi Acara Lambda	Fungsi Web Lambda	MicroVMS Lambda
Terbaik untuk	Request-response atau beban kerja berbasis peristiwa (API, pemrosesan data, otomatisasi)	Aplikasi web dan backend API yang melayani lalu lintas HTTP secara langsung	Lingkungan persisten yang menjalankan kode pengguna atau AI-produc ed tidak tepercaya
Model pemrogram an	Fungsi handler dipanggil dalam runtime yang didukung	Server HTTP yang mendengarkan pada port, menggunakan kerangka kerja web standar	Aplikasi apa pun - jalankan binari Anda sendiri, dengarkan port, gunakan kemampuan OS Linux

	Fungsi Acara Lambda	Fungsi Web Lambda	MicroVMS Lambda
Durasi	Hingga 15 menit per pemanggilan secara default; hingga 90 menit per pemanggil an pemetaan sumber asinkron atau sumber peristiwa pada Instans Terkelola Lambda; alur kerja multi-langkah yang berlangsung hingga satu tahun dengan Fungsi Tahan Lama Lambda	Hingga 15 menit per permintaan HTTP (default 30 detik)	Hingga 8 jam per sesi; menangguhkan dan melanjutkan di seluruh sesi
Lingkunga n runtime	Service-provided runtime bahasa; dukungan untuk runtime yang disediakan pelanggan	Service-provided Node.jsruntime	Customer-provided Gambar MicroVM
Jaringan Inbound	Pemanggilan langsung atau integrasi sumber peristiwa dengan layanan AWS	Service-provided Titik akhir HTTPS dengan TLS; perutean berbasis latensi opsional di seluruh Wilayah AWS	Akses masuk ke port apa pun menggunakan protokol OSI Layer 7
Konkurensi	Satu permintaan per lingkungan eksekusi pada satu waktu secara default; beberapa permintaan bersamaan per lingkungan eksekusi untuk Instans Terkelola Lambda	Beberapa permintaan bersamaan per lingkunga n eksekusi	Beberapa koneksi bersamaan per microVM

	Fungsi Acara Lambda	Fungsi Web Lambda	MicroVMS Lambda
Lingkungan Negara	Lingkungan eksekusi dapat digunakan kembali (mulai hangat), tetapi status mungkin tidak bertahan di seluruh pemanggilan	Lingkungan eksekusi dapat digunakan kembali (mulai hangat), tetapi status mungkin tidak bertahan di seluruh pemanggilan	Memori dan status disk dipertahankan pada penangguhan; dipulihkan pada resume
Penskalaan	Otomatis — Lambda menciptakan dan menghancurkan lingkungan eksekusi sebagai respons terhadap lalu lintas	Otomatis — Lambda menciptakan dan menghancurkan lingkungan eksekusi sebagai respons terhadap lalu lintas, menskalakan ke nol saat idle	Developer-controlled — Anda membuat, menangguhkan, melanjutkan, dan menghentikan microVMS melalui API
Siklus hidup	Sepenuhnya dikelola oleh Lambda	Sepenuhnya dikelola oleh Lambda	Developer-controlled; kebijakan idle opsional untuk resume suspend-otomatis
Harga	Per-request + GB-seconds waktu eksekusi	Per permintaan, ditambah waktu CPU aktif (ditagih dalam kenaikan 1 ms) dan memori untuk masa pakai setiap lingkungan eksekusi	Per-second komputasi saat menjalankan + penyimpanan snapshot saat ditangguhkan

Panduan penyiapan agen

Agen pengkodean AI dapat mempercepat pengembangan tanpa server dengan memberikan bantuan cerdas untuk perancah aplikasi, konfigurasi penerapan, debugging, dan pembuatan infrastruktur-sebagai-kode. Dengan mengonfigurasi agen Anda dengan kombinasi plugin dan keterampilan yang tepat, Anda melengkapinya dengan keahlian yang diperlukan untuk membangun dan mengelola aplikasi tanpa server.

Bekerja dengan agen pengkodean Anda

Untuk referensi mulai cepat, Anda dapat memasukkan tautan ini langsung ke konteks agen Anda:

```
https://docs.aws.amazon.com/lambda/latest/dg/samples/aws-lambda-agent-setup.md
```

Pilih metode instalasi yang sesuai dengan lingkungan pengembangan Anda.

Prasyarat

Pastikan prasyarat berikut tersedia untuk menginstal plugin aws-core:

- uv diinstal pada sistem Anda.
- (Opsional) Sebuah Akun AWS dengan kredensyal IAM yang disiapkan di mesin lokal Anda. Kredensyal diperlukan untuk alat yang menjalankan panggilan AWS API dan menjalankan skrip, tetapi tidak untuk mencari dokumentasi atau menemukan keterampilan. Jika Anda tidak memiliki kredensyal yang dikonfigurasi, lihat Menyiapkan Server AWS MCP untuk petunjuk terperinci.

Kode Claude

Instal plugin aws-core, bagian dari Agent Toolkit untuk AWS, yang menggabungkan konfigurasi AWS MCP Server dan serangkaian keterampilan agen yang dikuratori (termasuk keterampilan tanpa server). AWS

Instal plugin aws-core dari Agen Toolkit untuk AWS

Jalankan perintah berikut untuk menginstal plugin:

```
/plugin marketplace add aws/agent-toolkit-for-aws
/plugin install aws-core@agent-toolkit-for-aws
```

Ini menginstal keterampilan agen termasuk keterampilan AWS tanpa server, kait agen, dan konfigurasi AWS MCP Server dalam satu langkah.

kodeks

Instal plugin aws-core dari Agen Toolkit untuk AWS

Jalankan perintah berikut untuk menginstal plugin:

```
codex plugin marketplace add aws/agent-toolkit-for-aws
```

Kemudian luncurkan Codex dan jalankan `/plugins` untuk menelusuri dan menginstal plugin aws-core.

Kursor

Menginstal AWS keterampilan tanpa server

```
npx skills add https://github.com/aws/agent-toolkit-for-aws --skill aws-serverless --
yes --global
```

Kiro

Menginstal AWS keterampilan tanpa server di Kiro CLI

```
npx skills add https://github.com/aws/agent-toolkit-for-aws --skill aws-serverless --
yes --global
```

Instal kekuatan Kiro (dari Kiro IDE)

Instal kekuatan Kiro berikut yang menyediakan konteks dan alat khusus untuk agen Kiro sesuai permintaan:

- Model Aplikasi Tanpa Server (SAM) - instal melalui tautan kekuatan Kiro.
- Instans Terkelola Lambda — instal melalui tautan kekuatan Kiro.

GitHub Kopilot

Menginstal AWS keterampilan tanpa server

```
npx skills add https://github.com/aws/agent-toolkit-for-aws --skill aws-serverless --
yes --global
```

Devin Desktop

Menginstal AWS keterampilan tanpa server

```
npx skills add https://github.com/aws/agent-toolkit-for-aws --skill aws-serverless --
yes --global
```

OpenCode

Menginstal AWS keterampilan tanpa server

```
npx skills add https://github.com/aws/agent-toolkit-for-aws --skill aws-serverless --
yes --global
```

Untuk semua agen lain yang kompatibel dengan keterampilan agen

Untuk agen lain yang mendukung format keterampilan agen sumber terbuka, ikuti langkah-langkah berikut:

Menginstal AWS keterampilan tanpa server

```
npx skills add https://github.com/aws/agent-toolkit-for-aws --skill aws-serverless --
yes --global
```

Instal keterampilan Lambda tambahan

Agen Toolkit untuk AWS mencakup keterampilan tanpa server khusus yang memberi agen AI Anda pengetahuan tentang kemampuan Lambda. Instal keterampilan ini menggunakan agen apa pun seperti Kode Claude, Kursor, Kiro CLI, GitHub Copilot, Codex, Devin Desktop, OpenCode atau agen lain yang kompatibel dengan format keterampilan agen sumber terbuka untuk memungkinkan agen Anda membantu Anda membangun, menyebarkan, dan mengelola beban kerja menggunakan Fungsi Web Lambda, MicroVMS Lambda, Instans Terkelola Lambda, dan fungsi tahan lama Lambda.

1. Keterampilan Fungsi Web Lambda — Fungsi Web Lambda memungkinkan Anda menjalankan aplikasi web dan backend API sebagai server HTTP, tempat Node.js kode Anda menangani permintaan secara langsung menggunakan kerangka kerja seperti,, Fastify atau, Express Hono dengan Lambda menyediakan titik akhir HTTPS, penskalaan otomatis (termasuk ke nol), dan replikasi Multi-wilayah.

   ```
   npx skills add https://github.com/aws/agent-toolkit-for-aws/tree/main/skills/
   specialized-skills/serverless-skills/aws-lambda-web-functions --yes --global
   ```

2. Keterampilan MicroVMS Lambda - Lambda MicroVMS adalah primitif komputasi tanpa server yang menyediakan VM-level isolasi, kecepatan peluncuran dan resume yang hampir instan, dan pelestarian status untuk mengeksekusi pengguna atau kode di lingkungan eksekusi yang terisolasi dan stateful. AI-generated

   ```
   npx skills add https://github.com/aws/agent-toolkit-for-aws/tree/main/skills/
   specialized-skills/serverless-skills/aws-lambda-microvms --yes --global
   ```

3. Keahlian Instans Terkelola Lambda — Dengan Instans Terkelola Lambda, Anda dapat menjalankan fungsi Lambda di instans Amazon EC2 sambil mempertahankan kesederhanaan operasional Lambda, sehingga Anda dapat mengakses konfigurasi komputasi khusus dan efisiensi biaya melalui keuntungan harga Amazon EC2 tanpa mengelola infrastruktur.

   ```
   npx skills add https://github.com/aws/agent-toolkit-for-aws/tree/main/skills/
   specialized-skills/serverless-skills/aws-lambda-managed-instances --yes --global
   ```

4. Keterampilan fungsi tahan lama Lambda — Fungsi Lambda yang tahan lama memungkinkan Anda membangun aplikasi yang tahan terhadap kesalahan dan berjalan lama yang dapat memeriksa kemajuan, pulih secara otomatis dari kegagalan, dan menangguhkan eksekusi hingga satu tahun saat menunggu tugas yang berjalan lama seperti manusia-in-the-loop proses.

```
npx skills add https://github.com/aws/agent-toolkit-for-aws/tree/main/skills/
specialized-skills/serverless-skills/aws-lambda-durable-functions --yes --global
```

AWS Lambda Fungsi Acara (sebelumnya fungsi Lambda)

> **Tip**
>
> Bergabunglah dengan pakar Tanpa Server untuk lokakarya langsung gratis untuk mempelajari cara membuat aplikasi Tanpa Server dengan praktik terbaik. Mendaftar untuk lokakarya tanpa server gratis di situs web Pengalaman. AWS

AWS Lambda Fungsi Acara, sebelumnya disebut fungsi Lambda, menjalankan kode Anda sebagai respons terhadap peristiwa dari sumber seperti Amazon Simple Storage Service (Amazon S3), Amazon Simple Queue Service (Amazon Simple Queue Service), dan, tanpa perlu mengelola server. EventBridge Fungsi menerima peristiwa, menjalankan handler Anda, dan mengembalikan respons. Fungsi menskalakan naik dan turun secara otomatis dengan harga bayar per penggunaan. Untuk memulai, lihat Buat fungsi pertama Anda. Untuk informasi tentang praktik terbaik, bahasa pemrograman, isolasi penyewa, dan alat untuk pengembangan dan penyebaran, lihat topik terkait di bawah ini.

> **Note**
>
> Fungsi Acara adalah nama untuk jenis fungsi Lambda asli, dan konsol Lambda memberi label fungsi Event. Sumber daya yang mendasarinya tidak berubah: operasi API (seperti `CreateFunction` dan`Invoke`), AWS SDK, AWS CLI, dan jenis `AWS::Lambda::Function` sumber daya masih menggunakan fungsi, jadi Anda tidak perlu mengubah kode atau konfigurasi apa pun.

Untuk informasi harga, lihat Harga AWS Lambda.

Kapan menggunakan Fungsi Acara

Anda dapat menggunakan Fungsi Acara untuk:

- Pemrosesan file: Memproses file secara otomatis saat diunggah ke Amazon Simple Storage Service. Lihat contoh pemrosesan file untuk detailnya.
- Long-running alur kerja: Bangun alur kerja multi-langkah stateful yang dapat berjalan hingga satu tahun. Lihat fungsi Lambda yang tahan lama untuk detailnya.

- Operasi database dan contoh integrasi: Menanggapi perubahan database dan mengotomatiskan alur kerja data. Lihat contoh database untuk detailnya.
- Tugas terjadwal dan berkala: Jalankan operasi otomatis pada jadwal reguler menggunakan EventBridge. Lihat contoh tugas terjadwal untuk detailnya.
- Pemrosesan aliran: Memproses aliran data waktu nyata untuk analitik dan pemantauan. Lihat Kinesis Data Streams untuk detailnya.
- Aplikasi web: Menangani permintaan web dan API melalui Amazon API Gateway, Application Load Balancer, atau URL fungsi Lambda. Secara default, setiap lingkungan eksekusi menangani satu permintaan pada satu waktu. Pada Instans Terkelola Lambda, lingkungan eksekusi menangani beberapa permintaan secara bersamaan.
- Backend seluler: Buat backend API yang aman untuk aplikasi seluler dan web.
- Backend IoT: Menangani permintaan API web, seluler, IoT, dan pihak ketiga. Lihat IoT untuk detailnya.

Untuk melayani lalu lintas HTTP langsung dari server web standar, gunakan Lambda Web Functions. Untuk informasi selengkapnya, lihat Fungsi Web.

Fungsi Acara dan fungsi tahan lama

Fungsi Lambda berjalan hingga 15 menit per pemanggilan secara default dan ideal untuk tugas berbasis peristiwa seperti memproses permintaan API, menangani unggahan file, atau menanggapi perubahan basis data. Pada Instans Terkelola Lambda, fungsi dapat berjalan hingga 90 menit untuk pemanggilan asinkron dan pemetaan sumber peristiwa (ESM). Fungsi tahan lama memperluas model ini untuk beban kerja yang perlu berjalan lebih lama dan bertahan dari gangguan. Mereka secara otomatis memeriksa kemajuan mereka sehingga mereka melanjutkan dengan andal setelah kegagalan, dan mereka dapat berjalan hingga satu tahun ketika dipanggil secara asinkron. Gunakan fungsi yang tahan lama saat Anda membutuhkan alur kerja multi-langkah, persetujuan manusia-in-the-loop, atau koordinasi lintas layanan selama periode yang lama.

Bagaimana Lambda bekerja

Saat menggunakan Lambda, Anda hanya bertanggung jawab atas kode Anda. Lambda menjalankan kode Anda pada infrastruktur komputasi ketersediaan tinggi dan mengelola semua sumber daya komputasi, termasuk pemeliharaan server dan sistem operasi, penyediaan kapasitas, penskalaan otomatis, dan pencatatan.

Karena Lambda adalah layanan komputasi tanpa server, berbasis peristiwa, ia menggunakan pendekatan pemrograman yang berbeda dari aplikasi web tradisional. Model berikut menggambarkan cara kerja Lambda:

1. Anda menulis dan mengatur kode Anda dalam fungsi Lambda, yang merupakan blok bangunan dasar yang Anda gunakan untuk membuat aplikasi Lambda.
2. Anda mengontrol keamanan dan akses melalui izin Lambda, menggunakan peran eksekusi untuk mengelola AWS layanan apa yang dapat berinteraksi dengan fungsi Anda dan kebijakan sumber daya apa yang dapat berinteraksi dengan kode Anda.
3. Sumber dan AWS layanan peristiwa memicu fungsi Lambda Anda, meneruskan data peristiwa dalam format JSON, yang diproses fungsi Anda (ini termasuk pemetaan sumber peristiwa).
4. Lambda menjalankan kode Anda dengan runtime khusus bahasa (seperti dan Node.js Python) di lingkungan eksekusi yang mengemas runtime, lapisan, dan ekstensi Anda.

> **Tip**
>
> Untuk mempelajari cara membuat solusi tanpa server, lihat Panduan Pengembang Tanpa Server.

Fitur utama

Konfigurasikan, kontrol, dan terapkan aplikasi aman:

- Variabel lingkunganmemodifikasi perilaku aplikasi tanpa penerapan kode baru.
- Versiuji fitur baru dengan aman sambil mempertahankan lingkungan produksi yang stabil.
- Lapisanmengoptimalkan penggunaan kembali dan pemeliharaan kode dengan berbagi komponen umum di berbagai fungsi.
- Penandatanganan kodemenegakkan kepatuhan keamanan dengan memastikan hanya kode yang disetujui mencapai sistem produksi.

Skala dan lakukan dengan andal:

- Kontrol konkurensi dan penskalaan secara tepat mengelola respons aplikasi dan pemanfaatan sumber daya selama lonjakan lalu lintas.

- SnapStartsecara signifikan mengurangi waktu mulai dingin. Lambda SnapStart dapat memberikan kinerja startup serendah sub-detik, biasanya tanpa perubahan pada kode Fungsi Acara Anda.
- Streaming responsmengoptimalkan kinerja fungsi dengan memberikan muatan besar secara bertahap untuk pemrosesan waktu nyata.
- Image kontainerfungsi paket dengan dependensi kompleks menggunakan alur kerja kontainer.

Connect dan integrasikan dengan mulus:

- Jaringan VPC mengamankan sumber daya sensitif dan layanan internal.
- Sistem fileintegrasi yang berbagi data persisten dan mengelola operasi stateful di seluruh pemanggilan fungsi.
- URL fungsibuat API dan titik akhir yang menghadap publik tanpa layanan tambahan.
- Ekstensimeningkatkan fungsi dengan alat pemantauan, keamanan, dan operasional.

Informasi Terkait

- Untuk informasi tentang cara kerja Lambda, lihat. Bagaimana Lambda bekerja
- Untuk mulai menggunakan Lambda, lihat. Buat fungsi Lambda pertama Anda
- Untuk daftar contoh aplikasi, lihatMemulai dengan contoh aplikasi dan pola.

Bagaimana Lambda bekerja

Fungsi Lambda adalah blok bangunan dasar yang Anda gunakan untuk membangun aplikasi Lambda. Untuk menulis fungsi, penting untuk memahami konsep inti dan komponen yang membentuk model pemrograman Lambda. Bagian ini memandu Anda melalui elemen dasar yang perlu Anda ketahui untuk mulai membangun aplikasi tanpa server dengan Lambda.

- Fungsi lambda dan penangan fungsi- Fungsi Lambda adalah blok kecil kode yang berjalan sebagai respons terhadap peristiwa. Fungsi adalah blok bangunan dasar yang Anda gunakan untuk membangun aplikasi. Function handler adalah titik masuk untuk objek peristiwa yang diproses kode fungsi Lambda Anda.
- Lingkungan eksekusi Lambda dan runtime- Lingkungan eksekusi Lambda mengelola sumber daya yang diperlukan untuk menjalankan fungsi Anda. Runtime adalah lingkungan khusus bahasa yang dijalankan fungsi Anda.

- Peristiwa dan pemicu- Lain Layanan AWS nya dapat memanggil fungsi Anda sebagai respons terhadap peristiwa tertentu.
- Izin dan peran Lambda- Kontrol siapa yang dapat mengakses fungsi Anda dan fungsi apa Layanan AWS lagi yang dapat berinteraksi dengan Anda.

> **Tip**
>
> Jika Anda ingin memulai dengan memahami pengembangan tanpa server secara lebih umum, lihat Mem ahami perbedaan antara pengembangan tradisional dan tanpa server di Panduan Pengembang AWS Tanpa Server.

Fungsi lambda dan penangan fungsi

Di Lambda, fungsi adalah blok bangunan dasar yang Anda gunakan untuk membuat aplikasi. Fungsi Lambda adalah sepotong kode yang berjalan sebagai respons terhadap peristiwa, seperti pengguna memilih tombol di situs web atau file yang diunggah ke bucket Amazon Simple Storage Service (Amazon S3). Dengan fungsi tahan lama, kode Anda dapat menjeda eksekusi di antara langkah-langkah, mempertahankan status secara otomatis, menjadikannya ideal untuk alur kerja yang berjalan lama seperti pemrosesan pesanan atau moderasi konten. Anda dapat menganggap fungsi sebagai semacam program mandiri dengan properti berikut.

Handler fungsi Lambda adalah metode dalam kode fungsi Anda yang memproses peristiwa. Ketika fungsi berjalan sebagai respons terhadap suatu peristiwa, Lambda menjalankan pengendali fungsi. Data tentang peristiwa yang menyebabkan fungsi berjalan diteruskan langsung ke handler. Sementara kode dalam fungsi Lambda dapat berisi lebih dari satu metode atau fungsi, fungsi Lambda hanya dapat memiliki satu handler.

Untuk membuat fungsi Lambda, Anda menggabungkan kode fungsi dan dependensinya dalam paket penerapan. Lambda mendukung dua jenis paket penyebaran, arsip fil e.zip dan gambar kontainer.

- Suatu fungsi memiliki satu pekerjaan atau tujuan tertentu
- Mereka berjalan hanya ketika diperlukan dalam menanggapi peristiwa tertentu
- Mereka secara otomatis berhenti berjalan setelah selesai

Lingkungan eksekusi Lambda dan runtime

Fungsi Lambda berjalan di dalam lingkungan eksekusi yang aman dan terisolasi yang dikelola Lambda untuk Anda. Untuk fungsi yang tahan lama, lingkungan eksekusi mencakup komponen tambahan untuk manajemen status dan koordinasi alur kerja. Lingkungan eksekusi mengelola proses dan sumber daya yang diperlukan untuk menjalankan fungsi Anda. Ketika fungsi pertama kali dipanggil, Lambda menciptakan lingkungan eksekusi baru untuk fungsi dijalankan. Setelah fungsi selesai berjalan, Lambda tidak menghentikan lingkungan eksekusi segera; jika fungsi dipanggil lagi, Lambda dapat menggunakan kembali lingkungan eksekusi yang ada.

Lingkungan eksekusi Lambda juga berisi runtime, lingkungan khusus bahasa yang menyampaikan informasi peristiwa dan respons antara Lambda dan fungsi Anda. Lambda menyediakan sejumlah runtime ter kelola untuk bahasa pemrograman paling populer, atau Anda dapat membuatnya sendiri.

Untuk runtime terkelola, Lambda secara otomatis menerapkan pembaruan keamanan dan patch ke fungsi yang menggunakan runtime.

Peristiwa dan pemicu

Anda juga dapat memanggil fungsi Lambda secara langsung dengan menggunakan konsol Lambda, AWS CLI, atau salah satu Perangkat Pengembangan Perangkat AWS Lunak (SDK). Ini lebih biasa dalam aplikasi produksi untuk fungsi Anda dipanggil oleh orang lain Layanan AWS sebagai respons terhadap peristiwa tertentu. Misalnya, Anda mungkin ingin fungsi berjalan setiap kali item ditambahkan ke tabel Amazon DynamoDB.

Untuk membuat fungsi Anda merespons peristiwa, Anda mengatur pem icu. Pemicu menghubungkan fungsi Anda ke sumber peristiwa, dan fungsi Anda dapat memiliki beberapa pemicu. Ketika suatu peristiwa terjadi, Lambda menerima data peristiwa sebagai dokumen JSON dan mengubahnya menjadi objek yang dapat diproses kode Anda. Anda dapat menentukan format JSON berikut untuk acara Anda dan runtime Lambda mengonversi JSON ini menjadi objek sebelum meneruskannya ke handler fungsi Anda.

Example acara Lambda kustom

```
{
  "Location": "SEA",
  "WeatherData":{
    "TemperaturesF":{
      "MinTempF": 22,
```

```
      "MaxTempF": 78
    },
    "PressuresHPa":{
      "MinPressureHPa": 1015,
      "MaxPressureHPa": 1027
    }
  }
}
```

Layanan streaming dan antrian seperti Amazon Kinesis atau Amazon SQS menggunakan pemetaan sumber peristiwa alih- alih pemicu standar. Pemetaan sumber peristiwa mensurvei sumber untuk data baru, mengumpulkan catatan bersama, dan kemudian memanggil fungsi Anda dengan peristiwa batch. Untuk informasi selengkapnya, lihat Bagaimana pemetaan sumber peristiwa berbeda dari pemicu langsung.

Untuk memahami cara kerja pemicu, mulailah dengan menyelesaikan tutorial Gunakan pemicu Amazon S3, atau untuk gambaran umum tentang penggunaan pemicu dan instruksi untuk membuat pemicu menggunakan konsol Lambda, lihatMengintegrasikan layanan lain.

Izin dan peran Lambda

Untuk Lambda, ada dua jenis izin utama yang perlu Anda konfigurasi:

- Izin yang dibutuhkan fungsi Anda untuk mengakses lainnya Layanan AWS
- Izin yang dibutuhkan pengguna lain dan Layanan AWS untuk mengakses fungsi Anda

Bagian berikut menjelaskan kedua jenis izin ini dan membahas praktik terbaik untuk menerapkan izin hak istimewa terendah.

Izin untuk fungsi untuk mengakses lainnya AWS sumber daya

Fungsi Lambda sering perlu mengakses AWS sumber daya lain dan melakukan tindakan pada mereka. Misalnya, fungsi mungkin membaca item dari tabel DynamoDB, menyimpan objek dalam bucket S3, atau menulis ke antrian Amazon SQS. Untuk memberikan fungsi izin yang mereka butuhkan untuk melakukan tindakan ini, Anda menggunakan peran eksekusi .

Peran eksekusi Lambda adalah jenis peran khusus AWS Identity and Access Management (IAM) https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles.html, identitas yang Anda buat di akun Anda yang memiliki izin khusus yang terkait dengannya yang ditentukan dalam kebijakan.

Setiap fungsi Lambda harus memiliki peran eksekusi, dan satu peran dapat digunakan oleh lebih dari satu fungsi. Ketika fungsi dipanggil, Lambda mengambil peran eksekusi fungsi dan diberikan izin untuk mengambil tindakan yang ditentukan dalam kebijakan peran.

Saat Anda membuat fungsi di konsol Lambda, Lambda secara otomatis membuat peran eksekusi untuk fungsi Anda. Kebijakan peran memberikan izin dasar fungsi Anda untuk menulis keluaran log ke Amazon CloudWatch Logs. Untuk memberikan izin fungsi Anda untuk melakukan tindakan pada AWS sumber daya lain, Anda perlu mengedit peran untuk menambahkan izin tambahan.

Cara termudah untuk menambahkan izin adalah dengan menggunakan kebijakan ter AWS kelola. Kebijakan terkelola dibuat dan dikelola oleh AWS dan memberikan izin untuk banyak kasus penggunaan umum. Misalnya, jika fungsi Anda melakukan operasi CRUD pada tabel DynamoDB, Anda dapat menambahkan AmazonDynamoDBFullAccess kebijakan ke peran Anda.

Izin untuk pengguna dan sumber daya lain untuk mengakses fungsi Anda

Untuk memberikan Layanan AWS izin lain untuk mengakses fungsi Lambda Anda, Anda menggunakan kebijakan berbasis sumber daya. Di IAM, kebijakan berbasis sumber daya dilampirkan ke sumber daya (dalam hal ini, fungsi Lambda Anda) dan menentukan siapa yang dapat mengakses sumber daya dan tindakan apa yang diizinkan untuk diambil.

Agar fungsi lain Layanan AWS dapat memanggil fungsi Anda melalui pemicu, kebijakan berbasis sumber daya fungsi Anda harus memberikan izin layanan tersebut untuk menggunakan tindakan tersebut. `lambda:InvokeFunction` Jika Anda membuat pemicu menggunakan konsol, Lambda secara otomatis menambahkan izin ini untuk Anda.

Untuk memberikan izin kepada AWS pengguna lain untuk mengakses fungsi Anda, Anda dapat mendefinisikannya dalam kebijakan berbasis sumber daya fungsi Anda dengan cara yang persis sama seperti untuk sumber daya lain Layanan AWS . Anda juga dapat menggunakan kebijakan berbasis identitas yang terkait dengan pengguna.

Praktik terbaik untuk izin Lambda

Saat Anda menetapkan izin menggunakan kebijakan IAM, praktik terbaik keamanan adalah hanya memberikan izin yang diperlukan untuk melakukan tugas. Ini dikenal sebagai prinsip hak istimewa paling kecil. Untuk mulai memberikan izin untuk fungsi Anda, Anda dapat memilih untuk menggunakan kebijakan ter AWS kelola. Kebijakan terkelola dapat menjadi cara tercepat dan termudah untuk memberikan izin untuk melakukan tugas, tetapi kebijakan tersebut mungkin juga menyertakan izin lain yang tidak Anda perlukan. Saat Anda beralih dari pengembangan awal melalui

pengujian dan produksi, kami sarankan Anda mengurangi izin menjadi hanya izin yang diperlukan dengan menentukan kebijakan yang dikelola pelanggan Anda sendiri.

Prinsip yang sama berlaku saat memberikan izin untuk mengakses fungsi Anda menggunakan kebijakan berbasis sumber daya. Misalnya, jika Anda ingin memberikan izin kepada Amazon S3 untuk memanggil fungsi Anda, praktik terbaik adalah membatasi akses ke bucket individual, atau bucket khususnya Akun AWS, daripada memberikan izin menyeluruh ke layanan S3.

Menjalankan kode dengan Lambda

Saat Anda menulis fungsi Lambda, Anda membuat kode yang berjalan di lingkungan tanpa server yang unik. Memahami bagaimana Lambda benar-benar menjalankan kode Anda melibatkan dua aspek utama: model pemrograman yang mendefinisikan bagaimana kode Anda berinteraksi dengan Lambda, dan siklus hidup lingkungan eksekusi yang menentukan bagaimana Lambda mengelola lingkungan runtime kode Anda.

Model pemrograman Lambda

Model pemrograman berfungsi sebagai seperangkat aturan umum tentang cara Lambda bekerja dengan kode Anda, terlepas dari apakah Anda menulis dengan Python, Java, atau bahasa lain yang didukung. Model pemrograman mencakup runtime dan handler Anda.

Untuk fungsi tanpa eksekusi tahan lama:

1. Lambda menerima acara.
2. Lambda menggunakan runtime untuk menyiapkan acara dalam format yang dapat digunakan kode Anda.
3. Runtime mengirimkan event yang diformat ke handler Anda.
4. Handler Anda memproses acara menggunakan kode yang telah Anda tulis.

Penting untuk model ini adalah handler, di mana Lambda mengirimkan peristiwa untuk diproses oleh kode Anda. Anggap saja sebagai titik masuk ke kode Anda. Ketika Lambda menerima acara, ia meneruskan acara ini dan beberapa informasi konteks ke handler Anda. Handler kemudian menjalankan kode Anda untuk memproses peristiwa ini - misalnya, mungkin membaca file saat diunggah ke Amazon S3, menganalisis gambar, atau memperbarui database. Setelah kode Anda selesai memproses suatu peristiwa, handler siap untuk memproses yang berikutnya.

Model eksekusi Lambda

Sementara model pemrograman mendefinisikan bagaimana Lambda berinteraksi dengan kode Anda, lingkungan Eksekusi adalah tempat Lambda benar-benar menjalankan fungsi Anda — ini adalah ruang komputasi yang aman dan terisolasi yang dibuat khusus untuk fungsi Anda.

Setiap lingkungan mengikuti siklus hidup:

1. Inisialisasi: Pengaturan lingkungan dan pemuatan kode

2. Pemanggilan: Eksekusi tunggal kode fungsi
3. Shutdown: Pembersihan lingkungan

Lingkungan ini menangani aspek-aspek penting dalam menjalankan fungsi Anda. Ini menyediakan fungsi Anda dengan memori dan `/tmp` direktori untuk penyimpanan sementara.

Memahami model pemrograman Lambda

Fungsi Acara Lambda berjalan hingga 15 menit per pemanggilan secara default. Pemanggilan asinkron dan pemetaan sumber peristiwa (ESM) pada Instans Terkelola Lambda dapat berjalan hingga 90 menit. Fungsi Tahan Lama dapat berjalan hingga satu tahun. Meskipun keduanya berbagi konsep inti, Fungsi Tahan Lama menambahkan kemampuan untuk alur kerja stateful yang berjalan lama.

Lambda menyediakan model pemrograman yang umum untuk semua runtime. Model pemrograman mendefinisikan antarmuka antara kode Anda dan sistem Lambda. Anda memberi tahu Lambda titik masuk ke fungsi Anda dengan mendefinisikan handler dalam konfigurasi fungsi. Runtime berlalu dengan objek ke handler yang berisi acara invokasi dan konteks, seperti nama fungsi dan ID permintaan.

Untuk Fungsi Tahan Lama, handler juga menerima DurableContext objek yang menyediakan:

- Kemampuan Checkpointing melalui step ()
- Tunggu manajemen negara melalui wait () dan wait ForCallback ()
- Persistensi status otomatis di antara pemanggilan

Setelah handler selesai memproses peristiwa pertama, runtime mengirimkan peristiwa lain. Untuk Fungsi Tahan Lama, handler dapat menjeda eksekusi antar langkah, dan Lambda secara otomatis menyimpan dan mengembalikan status saat fungsi dilanjutkan. Kelas fungsi tetap di dalam memori, sehingga klien dan variabel yang dinyatakan di luar metode handler dalam kode inisialisasi dapat digunakan kembali. Untuk menghemat waktu pemrosesan pada peristiwa berikutnya, buat sumber daya yang dapat digunakan kembali seperti klien AWS SDK selama inisialisasi. Setelah diinisialisasi, setiap instans fungsi Anda dapat memproses ribuan permintaan.

Fungsi Anda juga memiliki akses ke penyimpanan lokal di `/tmp` direktori, cache sementara yang dapat digunakan untuk beberapa pemanggilan. Untuk informasi selengkapnya, lihat Lingkungan eksekusi.

Saat AWS X-Ray pelacakan diaktifkan, runtime mencatat subsegmen terpisah untuk inisialisasi dan eksekusi.

Runtime menangkap output logging dari fungsi Anda dan mengirimkannya ke Amazon CloudWatch Logs. Selain mencatat output fungsi Anda, runtime juga mencatat entri saat invokasi fungsi dimulai dan berakhir. Ini termasuk catatan laporan dengan ID permintaan, durasi yang ditagih, durasi inisialisasi, dan perincian lainnya. Jika fungsi Anda menampilkan kesalahan, runtime mengembalikan kesalahan tersebut ke pemanggil.

> **Note**
>
> Logging tunduk pada kuota CloudWatch Log. Data log dapat hilang karena pelambatan atau, dalam beberapa kasus, ketika satu instans fungsi Anda dihentikan.

Perbedaan utama untuk Fungsi Tahan Lama:

- Status secara otomatis bertahan di antara langkah-langkah
- Fungsi dapat menjeda eksekusi tanpa menghabiskan sumber daya
- Langkah-langkah secara otomatis dicoba ulang pada kegagalan
- Kemajuan dilacak melalui pos pemeriksaan

Lambda memperluas fungsi Anda dengan menjalankan instans tambahannya seiring peningkatan permintaan, dan dengan menghentikan instans seiring dengan penurunan permintaan. Model ini mengarah pada variasi dalam arsitektur aplikasi, seperti:

- Kecuali dinyatakan sebaliknya, permintaan masuk dapat diproses di luar pesanan atau bersamaan.
- Jangan mengandalkan contoh fungsi Anda yang berumur panjang, alih-alih simpan status aplikasi Anda di tempat lain.
- Gunakan penyimpanan lokal dan objek tingkat kelas untuk meningkatkan performa, tetapi jaga ukuran paket deployment Anda seminimum mungkin dan jumlah data yang Anda transfer ke lingkungan eksekusi.

Untuk pengenalan langsung model pemrograman dalam bahasa pemrograman pilihan Anda, lihat bab berikut.

- Membangun Fungsi Acara Lambda dengan Node.js

- Membangun Fungsi Acara Lambda dengan Python
- Membangun Fungsi Acara Lambda dengan Ruby
- Membangun Fungsi Acara Lambda dengan Java
- Membangun Fungsi Acara Lambda dengan Go
- Membangun Fungsi Acara Lambda dengan C #
- Membangun Fungsi Acara Lambda dengan PowerShell

Memahami siklus hidup lingkungan pelaksanaan Lambda

Lingkungan eksekusi Lambda menjalankan pemanggilan fungsi hingga 15 menit secara default, dan fungsi tahan lama dapat berjalan hingga satu tahun. Fungsi tahan lama berbagi siklus hidup dasar yang sama dan menambahkan kemampuan manajemen status untuk alur kerja yang berjalan lama.

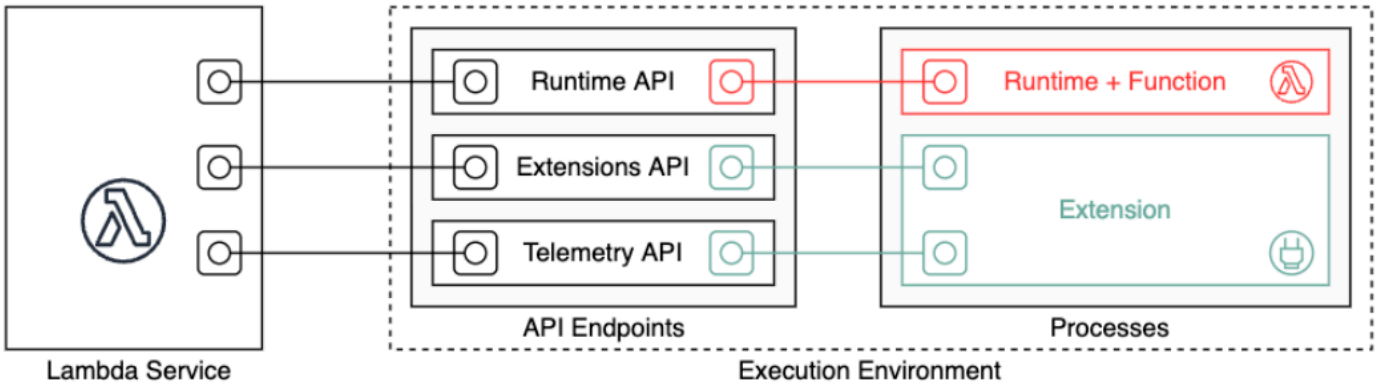
Lambda melakukan invokasi fungsi dalam lingkungan eksekusi, yang menyediakan lingkungan waktu pengoperasian yang aman dan terisolasi. Lingkungan eksekusi mengelola sumber daya yang diperlukan untuk menjalankan fungsi Anda. Lingkungan eksekusi juga menyediakan dukungan siklus hidup untuk waktu pengoperasian fungsi dan ekstensi eksternal terkait fungsi Anda.

Untuk Fungsi Tahan Lama, lingkungan eksekusi mencakup komponen tambahan untuk:

- Nyatakan ketekunan di antara langkah-langkah
- Manajemen Checkpointing
- Tunggu koordinasi negara
- Pelacakan kemajuan

> **Lingkungan eksekusi Instans Terkelola Lambda**
>
> Jika Anda menggunakan Instans Terkelola Lambda, lingkungan eksekusi memiliki perbedaan penting dibandingkan dengan fungsi Lambda (default). Instans Terkelola mendukung pemanggilan bersamaan, menggunakan model siklus hidup yang berbeda, dan berjalan di infrastruktur milik pelanggan. Untuk informasi rinci tentang lingkungan eksekusi Instans Terkelola, lihatMemahami lingkungan eksekusi Instans Terkelola Lambda.

Waktu pengoperasian fungsi tersebut berkomunikasi dengan Lambda menggunakan API Waktu Pengoperasian. Ekstensi berkomunikasi dengan Lambda menggunakan API Ekstensi. Ekstensi juga dapat menerima pesan log dan telemetri lain dari fungsi dengan menggunakan API Telemetri.



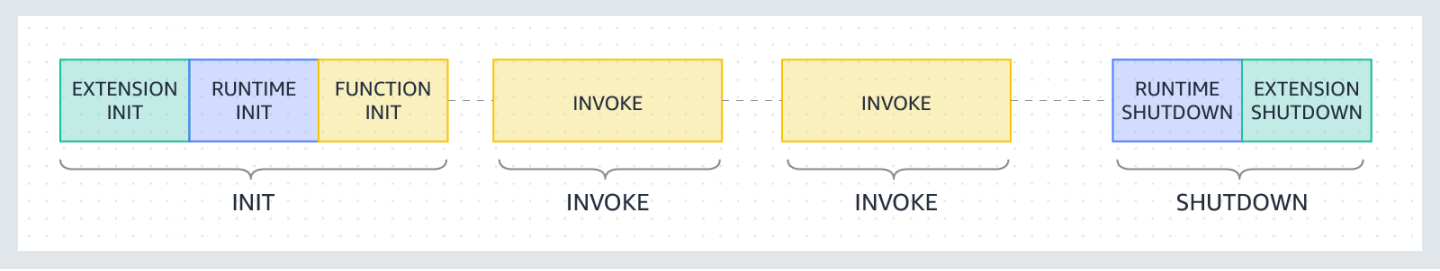
Saat membuat fungsi Lambda, Anda menentukan informasi konfigurasi, seperti jumlah memori yang tersedia dan waktu eksekusi maksimal yang diizinkan untuk fungsi Anda. Lambda menggunakan informasi ini untuk menyiapkan lingkungan eksekusi.

Waktu pengoperasian fungsi dan setiap ekstensi eksternal adalah proses yang berjalan di dalam lingkungan eksekusi. Izin, sumber daya, kredensial, dan variabel lingkungan dibagikan di antara fungsi dan ekstensi.

Topik

- Siklus hidup lingkungan eksekusi Lambda
- Mulai dingin dan latensi
- Mengurangi dingin dimulai dengan Provisioned Concurrency
- Mengoptimalkan inisialisasi statis

Siklus hidup lingkungan eksekusi Lambda



Setiap fase dimulai dengan kejadian yang dikirimkan Lambda ke waktu pengoperasian dan ke semua ekstensi terdaftar. Waktu pengoperasian dan setiap ekstensi menunjukkan penyelesaian dengan mengirimkan permintaan API `Next`. Lambda membekukan lingkungan eksekusi saat waktu pengoperasian dan setiap ekstensi telah selesai dan tidak ada kejadian yang tertunda.

Fase siklus hidup untuk Fungsi Tahan Lama meliputi:

- Init: Inisialisasi standar ditambah pengaturan status tahan lama
- Invoke: Dapat menyertakan eksekusi beberapa langkah dengan pos pemeriksaan otomatis
- Tunggu: Fungsi dapat menjeda eksekusi tanpa menghabiskan sumber daya
- Lanjutkan: Fungsi dimulai ulang dari pos pemeriksaan terakhir
- Shutdown: Pembersihan status dan sumber daya yang tahan lama

Topik

- Fase inisialisasi
- Kegagalan selama fase Init
- Fase pemulihan (hanya Lambda SnapStart)
- Fase invokasi
- Kegagalan selama fase pemanggilan
- Fase pematian

Fase inisialisasi

Di fase `Init`, Lambda melakukan tiga tugas:

- Memulai semua ekstensi (`Extension init`)
- Melakukan bootstrap waktu pengoperasian (`Runtime init`)
- Menjalankan kode statis fungsi (`Function init`)
- Jalankan kait runtime sebelum pos pemeriksaan (hanya Lambda) SnapStart

Fase `Init` berakhir ketika waktu pengoperasian dan semua ekstensi menandakan bahwa ekstensi siap dengan mengirim permintaan API `Next`. Fase `Init` dibatasi 10 detik. Jika ketiga tugas tidak selesai dalam 10 detik, Lambda mencoba kembali `Init` fase pada saat pemanggilan fungsi pertama dengan batas waktu fungsi yang dikonfigurasi.

Ketika SnapStart diaktifkan, `Init` fase terjadi ketika Anda mempublikasikan versi fungsi. Lambda menyimpan snapshot memori dan status disk dari lingkungan eksekusi yang diinisialisasi, mempertahankan snapshot terenkripsi, dan menyimpannya dalam cache untuk akses latensi rendah. Jika Anda memiliki hook runtime sebelum-checkpoint, maka kode berjalan di akhir fase. `Init`

> **Note**
>
> Batas waktu 10 detik tidak berlaku untuk fungsi yang menggunakan konkurensi yang disediakan,, SnapStart atau Instans Terkelola Lambda. Untuk konkurensi yang disediakan SnapStart, dan fungsi Instans Terkelola, kode inisialisasi Anda dapat berjalan hingga 15 menit. Batas waktu adalah 130 detik atau batas waktu fungsi yang dikonfigurasi (maksimum 900 detik), mana yang lebih tinggi.

Saat Anda menggunakan konkurensi yang disediakan, Lambda menginisialisasi lingkungan eksekusi saat Anda mengonfigurasi pengaturan PC untuk suatu fungsi. Lambda juga memastikan bahwa lingkungan eksekusi yang diinisialisasi selalu tersedia sebelum pemanggilan. Anda mungkin melihat kesenjangan antara fase pemanggilan dan inisialisasi fungsi Anda. Bergantung pada runtime dan konfigurasi memori fungsi Anda, Anda mungkin juga melihat latensi variabel pada pemanggilan pertama pada lingkungan eksekusi yang diinisialisasi.

Untuk fungsi yang menggunakan konkurensi sesuai permintaan, Lambda terkadang dapat menginisialisasi lingkungan eksekusi sebelum permintaan pemanggilan. Ketika ini terjadi, Anda mungkin juga mengamati kesenjangan waktu antara fase inisialisasi dan pemanggilan fungsi Anda. Kami menyarankan Anda untuk tidak mengambil ketergantungan pada perilaku ini.

Kegagalan selama fase Init

Jika fungsi macet atau habis waktu selama `Init` fase, Lambda memancarkan informasi kesalahan di log. `INIT_REPORT`

Example— Log INIT_REPORT untuk batas waktu

```
INIT_REPORT Init Duration: 1236.04 ms Phase: init Status: timeout
```

Example— Log INIT_REPORT untuk kegagalan ekstensi

```
INIT_REPORT Init Duration: 1236.04 ms Phase: init Status: error Error Type:
 Extension.Crash
```

Jika `Init` fase berhasil, Lambda tidak memancarkan `INIT_REPORT` log kecuali SnapStartatau konkurensi yang disediakan diaktifkan. SnapStart dan fungsi konkurensi yang disediakan selalu memancarkan. `INIT_REPORT` Untuk informasi selengkapnya, lihat Pemantauan untuk Lambda SnapStart.

Fase pemulihan (hanya Lambda SnapStart)

Saat Anda pertama kali memanggil SnapStartfungsi dan saat fungsi meningkat, Lambda melanjutkan lingkungan eksekusi baru dari snapshot yang bertahan alih-alih menginisialisasi fungsi dari awal. Jika Anda memiliki hook runtime after-restore, kode berjalan di akhir fase. `Restore` Anda dikenakan biaya untuk durasi kait runtime setelah pemulihan. Runtime harus dimuat dan kait runtime setelah pemulihan harus selesai dalam batas waktu tunggu (10 detik). Jika tidak, Anda akan mendapatkan SnapStartTimeoutException. Ketika `Restore` fase selesai, Lambda memanggil fungsi handler (the). Fase invokasi

Kegagalan selama fase Restore

Jika `Restore` fase gagal, Lambda memancarkan informasi kesalahan di log. `RESTORE_REPORT`

Example— RESTORE_REPORT log untuk batas waktu

```
RESTORE_REPORT Restore Duration: 1236.04 ms Status: timeout
```

Example— RESTORE_REPORT log untuk kegagalan hook runtime

```
RESTORE_REPORT Restore Duration: 1236.04 ms Status: error Error Type: Runtime.ExitError
```

Untuk informasi lebih lanjut tentang `RESTORE_REPORT` log, lihatPemantauan untuk Lambda SnapStart.

Fase invokasi

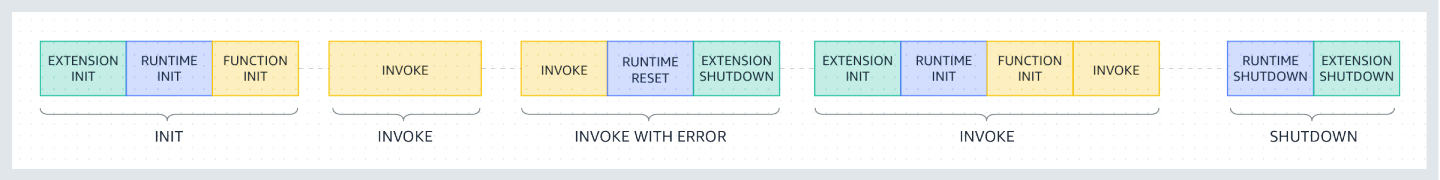
Saat fungsi Lambda diinvokasi untuk menanggapi permintaan API `Next`, Lambda mengirim kejadian `Invoke` ke waktu pengoperasian dan untuk setiap perpanjangan.

Pengaturan waktu habis fungsi membatasi durasi seluruh fase `Invoke`. Misalnya, jika Anda mengatur waktu fungsi habis sebagai 360 detik, fungsi dan semua ekstensi harus selesai dalam 360 detik. Perhatikan bahwa tidak ada fase pasca-invokasi yang independen. Durasi adalah jumlah semua waktu invokasi (waktu pengoperasian + ekstensi) dan tidak dihitung hingga fungsi dan semua ekstensi selesai dijalankan.

Fase invokasi berakhir ketika waktu pengoperasian dan semua ekstensi menandakan bahwa ekstensi selesai dengan mengirim permintaan API `Next`.

Kegagalan selama fase pemanggilan

Jika fungsi Lambda mengalami crash atau waktu habis selama fase `Invoke`, Lambda mengatur ulang lingkungan eksekusi. Diagram berikut menggambarkan perilaku lingkungan eksekusi Lambda ketika ada kegagalan pemanggilan:



Pada diagram sebelumnya:

- Fase pertama adalah fase INIT, yang berjalan tanpa kesalahan.
- Fase kedua adalah fase INVOKE, yang berjalan tanpa kesalahan.
- Pada titik tertentu, misalkan fungsi Anda mengalami kegagalan pemanggilan (penyebab umum termasuk batas waktu fungsi, kesalahan runtime, kelelahan memori, masalah konektivitas VPC, kesalahan izin, batas konkurensi, dan berbagai masalah konfigurasi). Untuk daftar lengkap kemungkinan kegagalan pemanggilan, lihat. the section called “Invokasi” Fase ketiga, berlabel INVOKE WITH ERROR, menggambarkan skenario ini.

  Ketika ini terjadi, layanan Lambda melakukan reset. Reset berperilaku seperti kejadian `Shutdown`. Pertama, Lambda mematikan runtime, lalu mengirimkan `Shutdown` acara ke setiap ekstensi eksternal terdaftar. Kejadian ini mencakup alasan untuk pematian. Jika lingkungan ini digunakan untuk pemanggilan baru, Lambda menginisialisasi ulang ekstensi dan runtime bersama dengan pemanggilan berikutnya.

  Perhatikan bahwa reset Lambda tidak menghapus konten `/tmp` direktori sebelum fase init berikutnya. Perilaku ini konsisten dengan fase shutdown reguler.

> **Note**
>
> AWS saat ini menerapkan perubahan pada layanan Lambda. Karena perubahan ini, Anda mungkin melihat perbedaan kecil antara struktur dan konten pesan log sistem dan segmen jejak yang dipancarkan oleh Fungsi Peristiwa Lambda yang berbeda di Anda. Akun AWS

Jika konfigurasi log sistem fungsi Anda disetel ke teks biasa, perubahan ini memengaruhi pesan log yang ditangkap di CloudWatch Log saat fungsi Anda mengalami kegagalan pemanggilan. Contoh berikut menunjukkan output log dalam format lama dan baru. Perubahan ini akan diterapkan selama beberapa minggu mendatang, dan semua fungsi di semua Wilayah AWS kecuali China GovCloud dan wilayah akan bertransisi untuk menggunakan pesan log format baru dan segmen pelacakan.

Example CloudWatch Keluaran log log (runtime atau ekstensi crash) - gaya lama

```
START RequestId: c3252230-c73d-49f6-8844-968c01d1e2e1 Version: $LATEST
RequestId: c3252230-c73d-49f6-8844-968c01d1e2e1 Error: Runtime exited without
 providing a reason
Runtime.ExitError
END RequestId: c3252230-c73d-49f6-8844-968c01d1e2e1
REPORT RequestId: c3252230-c73d-49f6-8844-968c01d1e2e1 Duration: 933.59 ms Billed
 Duration: 934 ms Memory Size: 128 MB Max Memory Used: 9 MB
```

Example CloudWatch Keluaran log log (batas waktu fungsi) - gaya lama

```
START RequestId: b70435cc-261c-4438-b9b6-efe4c8f04b21 Version: $LATEST
2024-03-04T17:22:38.033Z b70435cc-261c-4438-b9b6-efe4c8f04b21 Task timed out after
 3.00 seconds
END RequestId: b70435cc-261c-4438-b9b6-efe4c8f04b21
REPORT RequestId: b70435cc-261c-4438-b9b6-efe4c8f04b21 Duration: 3004.92 ms Billed
 Duration: 3117 ms Memory Size: 128 MB Max Memory Used: 33 MB Init Duration: 111.23
 ms
```

Format baru untuk CloudWatch log mencakup `status` bidang tambahan di `REPORT` baris. Dalam kasus runtime atau ekstensi crash, `REPORT` baris juga menyertakan bidang`ErrorType`.

Example CloudWatch Keluaran log log (runtime atau ekstensi crash) - gaya baru

```
START RequestId: 5b866fb1-7154-4af6-8078-6ef6ca4c2ddd Version: $LATEST
END RequestId: 5b866fb1-7154-4af6-8078-6ef6ca4c2ddd
REPORT RequestId: 5b866fb1-7154-4af6-8078-6ef6ca4c2ddd Duration: 133.61 ms Billed
 Duration: 214 ms Memory Size: 128 MB Max Memory Used: 31 MB Init Duration: 80.00
 ms Status: error Error Type: Runtime.ExitError
```

Example CloudWatch Keluaran log log (batas waktu fungsi) - gaya baru

```
START RequestId: 527cb862-4f5e-49a9-9ae4-a7edc90f0fda Version: $LATEST
END RequestId: 527cb862-4f5e-49a9-9ae4-a7edc90f0fda
REPORT RequestId: 527cb862-4f5e-49a9-9ae4-a7edc90f0fda Duration: 3016.78 ms Billed
 Duration: 3101 ms Memory Size: 128 MB Max Memory Used: 31 MB Init Duration: 84.00
 ms Status: timeout
```

- Fase keempat mewakili fase INVOKE segera setelah kegagalan pemanggilan. Di sini, Lambda menginisialisasi lingkungan lagi dengan menjalankan kembali fase INIT. Ini disebut init yang ditekan.

  Ketika input yang ditekan terjadi, Lambda tidak secara eksplisit melaporkan fase INIT tambahan di Log. CloudWatch Sebagai gantinya, Anda mungkin memperhatikan bahwa durasi di baris LAPORAN menyertakan durasi INIT tambahan+durasi INVOKE. Misalnya, Anda melihat log berikut di CloudWatch:

  ```
  2022-12-20T01:00:00.000-08:00 START RequestId: XXX Version: $LATEST
  2022-12-20T01:00:02.500-08:00 END RequestId: XXX
  2022-12-20T01:00:02.500-08:00 REPORT RequestId: XXX Duration: 3022.91 ms
  Billed Duration: 3000 ms Memory Size: 512 MB Max Memory Used: 157 MB
  ```

  Dalam contoh ini, perbedaan antara stempel waktu REPORT dan START adalah 2,5 detik. Ini tidak cocok dengan durasi yang dilaporkan 3022,91 milldetik, karena tidak memperhitungkan INIT ekstra (init yang ditekan) yang dilakukan Lambda. Dalam contoh ini, Anda dapat menyimpulkan bahwa fase INVOKE yang sebenarnya membutuhkan waktu 2,5 detik.

  Untuk wawasan lebih lanjut tentang perilaku ini, Anda dapat menggunakanMengakses data telemetri real-time untuk ekstensi menggunakan API Telemetri. API Telemetri memancarkan`INIT_START`,`INIT_RUNTIME_DONE`, dan `INIT_REPORT` peristiwa dengan input yang ditekan `phase=invoke` setiap kali terjadi di antara fase pemanggilan.
- Fase kelima mewakili fase SHUTDOWN, yang berjalan tanpa kesalahan.

Fase pematian

Ketika Lambda akan mematikan runtime, ia mengirimkan `Shutdown` acara ke setiap ekstensi eksternal yang terdaftar. Ekstensi dapat menggunakan waktu ini untuk tugas pembersihan akhir. Kejadian `Shutdown` adalah respons terhadap permintaan API `Next`.

Batas durasi: Durasi maksimum fase `Shutdown` bergantung pada konfigurasi ekstensi terdaftar:

- 0 ms - Fungsi tanpa ekstensi terdaftar
- 500 md – Fungsi dengan ekstensi internal terdaftar.
- 2.000 md – Fungsi dengan satu atau beberapa ekstensi eksternal terdaftar.

Jika waktu pengoperasian atau ekstensi tidak merespons kejadian `Shutdown` dalam batasan, Lambda mengakhiri proses menggunakan sinyal `SIGKILL`.

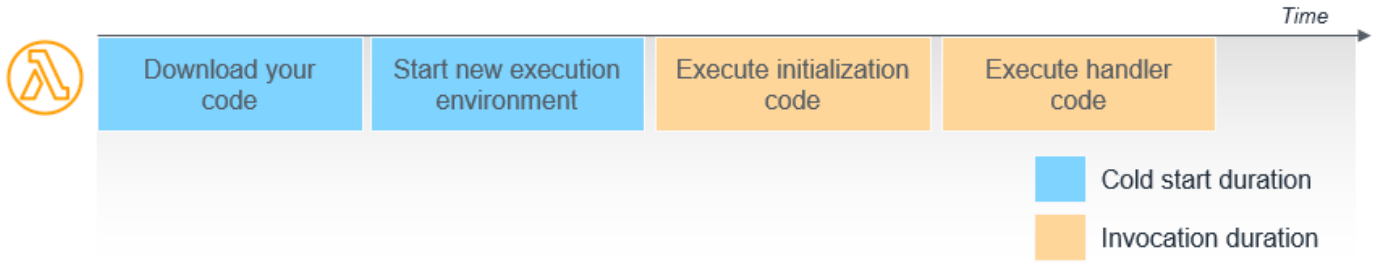
Setelah fungsi dan semua ekstensi selesai, Lambda mempertahankan lingkungan eksekusi selama beberapa waktu untuk mengantisipasi invokasi fungsi lain. Namun, Lambda menghentikan lingkungan eksekusi setiap beberapa jam untuk memungkinkan pembaruan dan pemeliharaan runtime — bahkan untuk fungsi yang dipanggil terus menerus. Anda tidak boleh berasumsi bahwa lingkungan eksekusi akan bertahan tanpa batas waktu. Untuk informasi selengkapnya, lihat Menerapkan tanpa kewarganegaraan dalam fungsi.

Saat fungsi dipanggil lagi, Lambda mencairkan lingkungan untuk digunakan kembali. Menggunakan kembali lingkungan eksekusi memiliki implikasi berikut:

- Objek yang dinyatakan di luar metode penangan fungsi tetap diinisialisasi, memberikan pengoptimalan tambahan ketika fungsi diinvokasi lagi. Misalnya, jika fungsi Lambda Anda menetapkan koneksi database, alih-alih menetapkan kembali koneksi, koneksi asli digunakan dalam invokasi berikutnya. Kami menyarankan Anda menambahkan logika dalam kode untuk memeriksa apakah ada koneksi sebelum membuat yang baru.
- Setiap lingkungan eksekusi menyediakan antara 512 MB dan 10.240 MB, dalam peningkatan 1-MB, ruang disk dalam direktori. `/tmp` Isi direktori tetap ada ketika lingkungan eksekusi dibekukan, menyediakan cache sementara yang dapat digunakan untuk beberapa invokasi. Anda dapat menambahkan kode tambahan untuk memeriksa apakah cache memiliki data yang Anda simpan. Untuk informasi lebih lanjut tentang batas ukuran deployment, lihat Kuota Lambda.
- Proses latar belakang atau panggilan balik yang diinisialisasi fungsi Lambda Anda dan tidak selesai ketika fungsi berakhir akan dilanjutkan jika Lambda menggunakan kembali lingkungan eksekusi. Pastikan setiap proses latar belakang atau panggilan balik dalam kode Anda selesai sebelum kode tersebut ditutup.

Mulai dingin dan latensi

Ketika Lambda menerima permintaan untuk menjalankan fungsi melalui API Lambda, layanan terlebih dahulu menyiapkan lingkungan eksekusi. Selama fase inisialisasi ini, layanan mengunduh kode Anda, memulai lingkungan, dan menjalankan kode inisialisasi apa pun di luar penangan utama. Akhirnya, Lambda menjalankan kode handler.



Dalam diagram ini, dua langkah pertama mengunduh kode dan mengatur lingkungan sering disebut sebagai “awal yang dingin”. Anda dikenakan biaya untuk kali ini, dan itu menambah latensi ke durasi pemanggilan Anda secara keseluruhan.

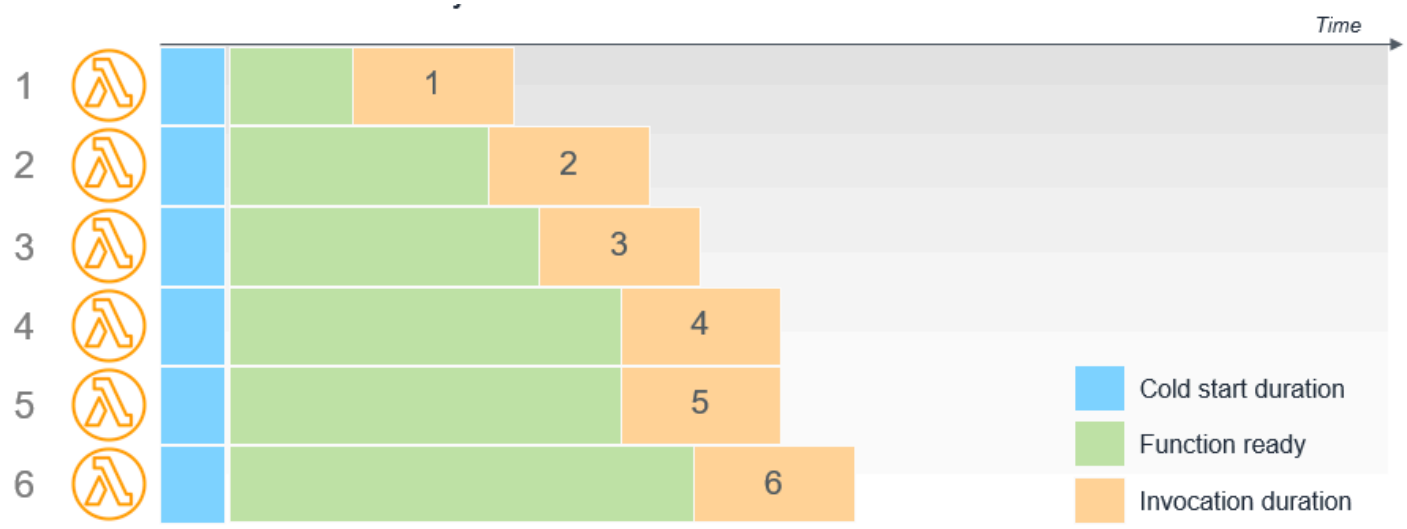
Setelah pemanggilan selesai, lingkungan eksekusi dibekukan. Untuk meningkatkan manajemen dan kinerja sumber daya, Lambda mempertahankan lingkungan eksekusi untuk jangka waktu tertentu. Selama waktu ini, jika permintaan lain tiba untuk fungsi yang sama, Lambda dapat menggunakan kembali lingkungan. Permintaan kedua ini biasanya selesai lebih cepat, karena lingkungan eksekusi sudah sepenuhnya diatur. Ini disebut “awal yang hangat”.

Permulaan dingin biasanya terjadi di bawah 1% dari doa. Durasi awal yang dingin bervariasi dari di bawah 100 ms hingga lebih dari 1 detik. Secara umum, start dingin biasanya lebih umum dalam fungsi pengembangan dan pengujian daripada beban kerja produksi. Ini karena fungsi pengembangan dan pengujian biasanya lebih jarang dipanggil.

Mengurangi dingin dimulai dengan Provisioned Concurrency

Jika Anda membutuhkan waktu mulai fungsi yang dapat diprediksi untuk beban kerja Anda, konkurensi yang disediakan adalah solusi yang disarankan untuk memastikan latensi serendah mungkin. Fitur ini melakukan pra-inisialisasi lingkungan eksekusi, mengurangi start dingin.

Misalnya, fungsi dengan konkurensi 6 yang disediakan memiliki 6 lingkungan eksekusi yang telah dipanaskan sebelumnya.



Mengoptimalkan inisialisasi statis

Inisialisasi statis terjadi sebelum kode handler mulai berjalan dalam suatu fungsi. Ini adalah kode inisialisasi yang Anda berikan, yang berada di luar penangan utama. Kode ini sering digunakan untuk mengimpor pustaka dan dependensi, mengatur konfigurasi, dan menginisialisasi koneksi ke layanan lain.

Contoh Python berikut menunjukkan mengimpor, dan mengonfigurasi modul, dan membuat klien Amazon S3 selama fase inisialisasi, sebelum fungsi berjalan selama pemanggilan. `lambda_handler`

```
import os
import json
import cv2
import logging
import boto3

s3 = boto3.client('s3')
logger = logging.getLogger()
logger.setLevel(logging.INFO)

def lambda_handler(event, context):

  # Handler logic...
```

Kontributor latensi terbesar sebelum eksekusi fungsi berasal dari kode inisialisasi. Kode ini berjalan ketika lingkungan eksekusi baru dibuat untuk pertama kalinya. Kode inisialisasi tidak dijalankan lagi

jika pemanggilan menggunakan lingkungan eksekusi yang hangat. Faktor-faktor yang mempengaruhi latensi kode inisialisasi meliputi:

- Ukuran paket fungsi, dalam hal pustaka dan dependensi yang diimpor, dan lapisan Lambda.
- Jumlah kode dan inisialisasi bekerja.
- Kinerja perpustakaan dan layanan lainnya dalam menyiapkan koneksi dan sumber daya lainnya.

Ada sejumlah langkah yang dapat diambil pengembang untuk mengoptimalkan latensi inisialisasi statis. Jika suatu fungsi memiliki banyak objek dan koneksi, Anda mungkin dapat merancang ulang satu fungsi menjadi beberapa fungsi khusus. Ini secara individual lebih kecil dan masing-masing memiliki kode inisialisasi yang lebih sedikit.

Penting bahwa fungsi hanya mengimpor pustaka dan dependensi yang mereka butuhkan. Misalnya, jika Anda hanya menggunakan Amazon DynamoDB di AWS SDK, Anda dapat meminta layanan individual alih-alih seluruh SDK. Bandingkan tiga contoh berikut:

```
// Instead of const AWS = require('aws-sdk'), use:
const DynamoDB = require('aws-sdk/clients/dynamodb')

// Instead of const AWSXRay = require('aws-xray-sdk'), use:
const AWSXRay = require('aws-xray-sdk-core')

// Instead of const AWS = AWSXRay.captureAWS(require('aws-sdk')), use:
const dynamodb = new DynamoDB.DocumentClient()
AWSXRay.captureAWSClient(dynamodb.service)
```

Inisialisasi statis juga sering merupakan tempat terbaik untuk membuka koneksi database untuk memungkinkan fungsi menggunakan kembali koneksi melalui beberapa pemanggilan ke lingkungan eksekusi yang sama. Namun, Anda mungkin memiliki sejumlah besar objek yang hanya digunakan di jalur eksekusi tertentu dalam fungsi Anda. Dalam hal ini, Anda dapat dengan malas memuat variabel dalam lingkup global untuk mengurangi durasi inisialisasi statis.

Hindari variabel global untuk informasi spesifik konteks. Jika fungsi Anda memiliki variabel global yang digunakan hanya untuk masa pakai pemanggilan tunggal dan disetel ulang untuk pemanggilan berikutnya, gunakan cakupan variabel yang bersifat lokal ke handler. Ini tidak hanya mencegah kebocoran variabel global di seluruh pemanggilan, tetapi juga meningkatkan kinerja inisialisasi statis.

Membuat arsitektur berbasis acara dengan Lambda

Peristiwa adalah apa pun yang memicu fungsi Lambda untuk dijalankan. Peristiwa dapat memicu fungsi Lambda dengan dua cara: melalui pemanggilan langsung (push) dan pemetaan sumber peristiwa (tarik).

Banyak AWS layanan dapat langsung menjalankan fungsi Lambda Anda. Layanan ini mendorong acara ke fungsi Lambda Anda. Peristiwa yang memicu fungsi dapat berupa apa saja, mulai dari permintaan HTTP melalui API Gateway, jadwal yang dikelola oleh EventBridge Scheduler, AWS IoT acara, atau peristiwa Amazon S3. Dengan pemetaan sumber peristiwa, Lambda secara aktif mengambil (atau menarik) peristiwa dari antrian atau aliran. Anda mengonfigurasi Lambda untuk memeriksa peristiwa dari layanan yang didukung, dan Lambda menangani polling dan pemanggilan fungsi Anda.

Ketika diteruskan ke fungsi Anda, peristiwa disusun dalam format JSON. Struktur JSON bervariasi tergantung pada layanan yang menghasilkannya dan jenis acara. Sementara pemanggilan fungsi Lambda dapat bertahan hingga 15 menit secara default, Lambda paling cocok untuk pemanggilan singkat yang berlangsung satu detik atau kurang. Hal ini terutama berlaku untuk arsitektur berbasis peristiwa, di mana setiap fungsi Lambda diperlakukan sebagai layanan mikro yang bertanggung jawab untuk melakukan serangkaian instruksi spesifik yang sempit.

> **Note**
>
> Event-driven arsitektur berkomunikasi di berbagai sistem menggunakan jaringan, yang memperkenalkan latensi variabel. Untuk beban kerja yang membutuhkan latensi sangat rendah, seperti sistem perdagangan real-time, desain ini mungkin bukan pilihan terbaik. Namun, untuk beban kerja yang sangat skalabel dan tersedia, atau yang memiliki pola lalu lintas yang tidak dapat diprediksi, arsitektur berbasis peristiwa dapat memberikan cara yang efektif untuk memenuhi tuntutan ini.

Topik

- Manfaat arsitektur berbasis peristiwa
- Trade-offs arsitektur yang digerakkan oleh peristiwa
- Anti-patterns dalam Lambda-based aplikasi berbasis peristiwa

Manfaat arsitektur berbasis peristiwa

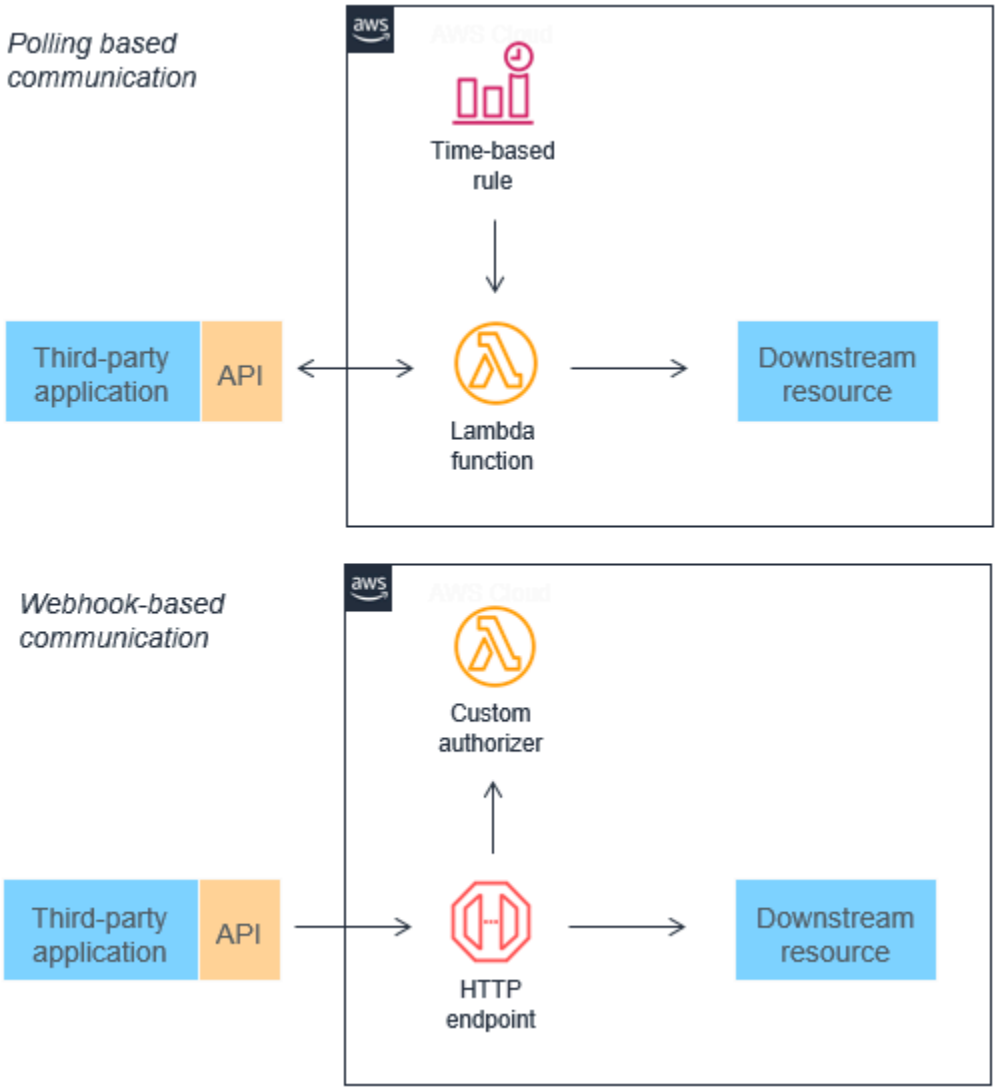
Lambda mendukung dua metode pemanggilan dalam arsitektur berbasis peristiwa:

1. Pemanggilan langsung (metode push): AWS layanan memicu fungsi Lambda secara langsung. Contoh:
   - Amazon S3 memicu fungsi saat file diunggah
   - API Gateway memicu fungsi saat menerima permintaan HTTP
2. Pemetaan sumber peristiwa (metode tarik): Lambda mengambil peristiwa dan memanggil fungsi. Contoh:
   - Lambda mengambil pesan dari antrian Amazon SQS dan memanggil fungsi
   - Lambda membaca catatan dari aliran DynamoDB dan memanggil fungsi

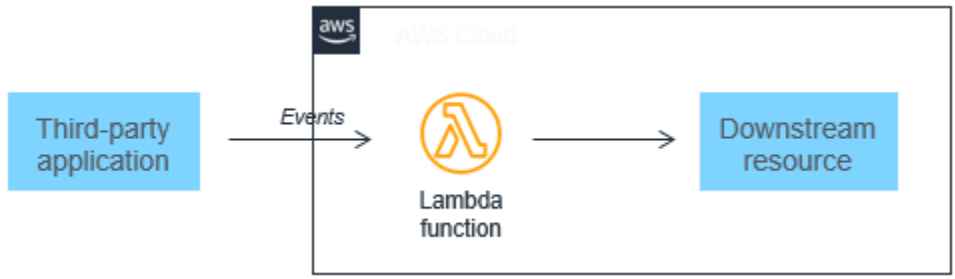
Kedua metode berkontribusi pada manfaat arsitektur berbasis peristiwa, seperti yang dijelaskan di bawah ini.

Mengganti polling dan webhook dengan acara

Banyak arsitektur tradisional menggunakan polling dan mekanisme webhook untuk mengkomunikasikan keadaan antara komponen yang berbeda. Polling bisa sangat tidak efisien untuk mengambil pembaruan karena ada jeda antara data baru yang tersedia dan sinkronisasi dengan layanan hilir. Webhook tidak selalu didukung oleh layanan mikro lain yang ingin Anda integrasikan. Mereka mungkin juga memerlukan otorisasi kustom dan konfigurasi otentikasi. Dalam kedua kasus, metode integrasi ini menantang untuk skala sesuai permintaan tanpa pekerjaan tambahan oleh tim pengembangan.



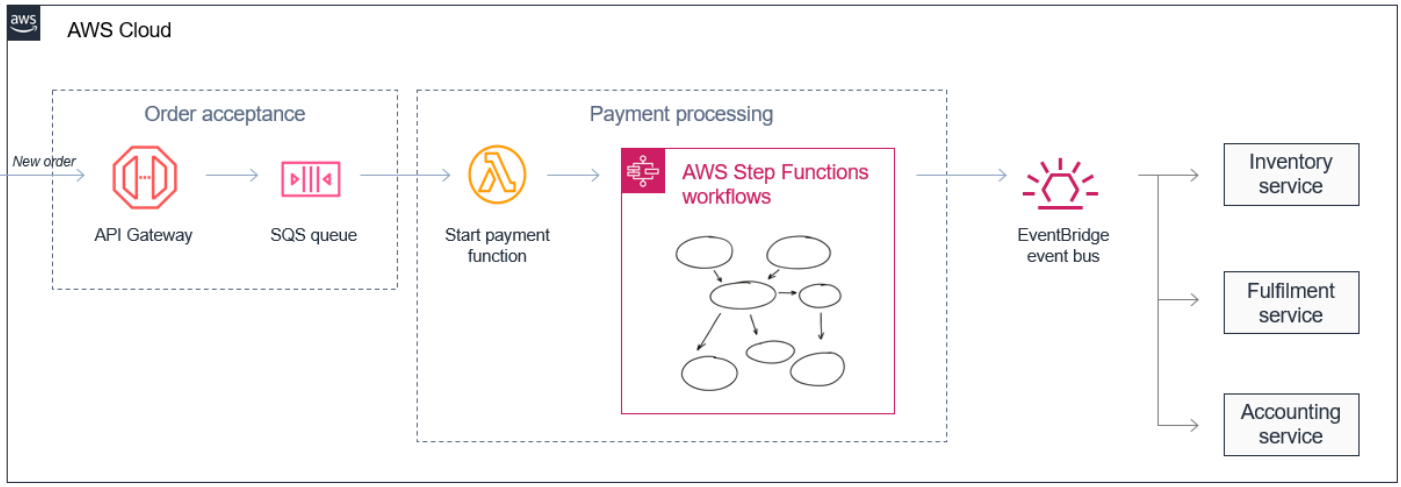
Kedua mekanisme ini dapat digantikan oleh peristiwa, yang dapat disaring, diarahkan, dan didorong ke hilir untuk mengkonsumsi layanan mikro. Pendekatan ini dapat menghasilkan konsumsi bandwidth yang lebih sedikit, pemanfaatan CPU, dan biaya yang berpotensi lebih rendah. Arsitektur ini juga dapat mengurangi kompleksitas, karena setiap unit fungsional lebih kecil dan seringkali ada lebih sedikit kode.



Event-driven Arsitektur juga dapat mempermudah merancang sistem yang mendekati waktu nyata, membantu organisasi menjauh dari pemrosesan berbasis batch. Peristiwa dihasilkan pada saat status dalam aplikasi berubah, sehingga kode kustom layanan mikro harus dirancang untuk menangani pemrosesan satu peristiwa. Karena penskalaan ditangani oleh layanan Lambda, arsitektur ini dapat menangani peningkatan lalu lintas yang signifikan tanpa mengubah kode kustom. Saat peristiwa meningkat, begitu juga lapisan komputasi yang memproses peristiwa.

Mengurangi kompleksitas

Microservices memungkinkan pengembang dan arsitek untuk menyederhanakan alur kerja yang kompleks. Misalnya, monolit e-niaga dapat dipecah menjadi penerimaan pesanan dan proses pembayaran dengan inventaris, pemenuhan, dan layanan akuntansi yang terpisah. Apa yang mungkin rumit untuk dikelola dan diatur dalam monolit menjadi serangkaian layanan terpisah yang berkomunikasi secara asinkron dengan peristiwa.



Pendekatan ini juga memungkinkan untuk merakit layanan yang memproses data pada tingkat yang berbeda. Dalam hal ini, layanan mikro penerimaan pesanan dapat menyimpan volume tinggi pesanan masuk dengan buffering pesan dalam antrian Amazon SQS.

Layanan pemrosesan pembayaran, yang biasanya lebih lambat karena kompleksitas penanganan pembayaran, dapat mengambil aliran pesan yang stabil dari antrian Amazon SQS. Ini dapat mengatur coba ulang yang kompleks dan logika penanganan kesalahan menggunakan AWS Step Functions, dan mengoordinasikan alur kerja pembayaran aktif untuk ratusan ribu pesanan.

Pendekatan alternatif: Untuk orkestrasi menggunakan bahasa pemrograman standar, Anda dapat menggunakan fungsi tahan lama Lambda. Fungsi tahan lama memungkinkan Anda menulis penerimaan pesanan, pemrosesan pembayaran, dan logika notifikasi dalam kode dengan pos

pemeriksaan otomatis dan coba lagi. Pendekatan ini bekerja dengan baik ketika alur kerja terutama melibatkan fungsi Lambda dan Anda lebih suka menyimpan logika orkestrasi dalam kode.

Meningkatkan skalabilitas dan ekstensibilitas

Layanan mikro menghasilkan peristiwa yang biasanya dipublikasikan ke layanan perpesanan seperti Amazon SNS dan Amazon SQS. Ini berperilaku seperti penyangga elastis antara layanan mikro dan membantu menangani penskalaan saat lalu lintas meningkat. Layanan seperti Amazon kemudian EventBridge dapat memfilter dan merutekan pesan tergantung pada konten acara, sebagaimana didefinisikan dalam aturan. Akibatnya, aplikasi berbasis peristiwa dapat lebih terukur dan menawarkan redundansi yang lebih besar daripada aplikasi monolitik.

Sistem ini juga sangat dapat diperluas, memungkinkan tim lain untuk memperluas fitur dan menambahkan fungsionalitas tanpa memengaruhi pemrosesan pesanan dan layanan mikro pemrosesan pembayaran. Dengan menerbitkan acara menggunakan EventBridge, aplikasi ini terintegrasi dengan sistem yang ada, seperti layanan mikro inventaris, tetapi juga memungkinkan aplikasi masa depan untuk diintegrasikan sebagai konsumen acara. Produsen acara tidak memiliki pengetahuan tentang konsumen acara, yang dapat membantu menyederhanakan logika layanan mikro.

Trade-offs arsitektur yang digerakkan oleh peristiwa

Latensi variabel

Tidak seperti aplikasi monolitik, yang mungkin memproses segala sesuatu dalam ruang memori yang sama pada satu perangkat, aplikasi berbasis peristiwa berkomunikasi di seluruh jaringan. Desain ini memperkenalkan latensi variabel. Meskipun dimungkinkan untuk merekayasa aplikasi untuk meminimalkan latensi, aplikasi monolitik hampir selalu dapat dioptimalkan untuk latensi yang lebih rendah dengan mengorbankan skalabilitas dan ketersediaan.

Beban kerja yang membutuhkan kinerja latensi rendah yang konsisten, seperti aplikasi perdagangan frekuensi tinggi di bank atau otomatisasi robotika sub-milidetik di gudang, bukanlah kandidat yang baik untuk arsitektur berbasis peristiwa.

Konsistensi akhirnya

Suatu peristiwa mewakili perubahan keadaan, dan dengan banyak peristiwa yang mengalir melalui layanan yang berbeda dalam arsitektur pada titik waktu tertentu, beban kerja seperti itu seringkali pada akhirnya konsisten. Ini membuatnya lebih kompleks untuk memproses transaksi, menangani duplikat, atau menentukan keadaan keseluruhan sistem yang tepat.

Beberapa beban kerja berisi kombinasi persyaratan yang pada akhirnya konsisten (misalnya, total pesanan dalam jam berjalan) atau sangat konsisten (misalnya, inventaris saat ini). Untuk beban kerja yang membutuhkan konsistensi data yang kuat, ada pola arsitektur untuk mendukung ini. Contoh:

- DynamoDB dapat memberikan pembacaan yang sangat konsisten, terkadang pada latensi yang lebih tinggi, mengkonsumsi throughput yang lebih besar daripada mode default. DynamoDB juga dapat mendukung transaksi untuk membantu menjaga konsistensi data.
- Anda dapat menggunakan Amazon RDS untuk fitur yang membutuhkan properti ACID, meskipun database relasional umumnya kurang skalabel daripada database NoSQL seperti DynamoDB. Amazon RDS Proxy dapat membantu mengelola pengumpulan dan penskalaan koneksi dari konsumen sementara seperti fungsi Lambda.

Event-based arsitektur biasanya dirancang di sekitar peristiwa individu, bukan kumpulan data yang besar. Umumnya, alur kerja dirancang untuk mengelola langkah-langkah peristiwa individu atau alur eksekusi alih-alih beroperasi pada beberapa peristiwa secara bersamaan. Dalam serverless, pemrosesan peristiwa real-time lebih disukai daripada pemrosesan batch: batch harus diganti dengan banyak pembaruan tambahan yang lebih kecil. Meskipun ini dapat membuat beban kerja lebih tersedia dan terukur, itu juga membuatnya lebih menantang bagi acara untuk memiliki kesadaran akan peristiwa lain.

Mengembalikan nilai ke penelepon

Dalam banyak kasus, aplikasi berbasis peristiwa bersifat asinkron. Ini berarti bahwa layanan penelepon tidak menunggu permintaan dari layanan lain sebelum melanjutkan pekerjaan lain. Ini adalah karakteristik mendasar dari arsitektur berbasis peristiwa yang memungkinkan skalabilitas dan fleksibilitas. Ini berarti bahwa meneruskan nilai pengembalian atau hasil alur kerja lebih kompleks daripada alur eksekusi sinkron.

Sebagian besar pemanggilan Lambda dalam sistem produksi bersifat asinkron, menanggapi peristiwa dari layanan seperti Amazon S3 atau Amazon SQS. Dalam kasus ini, keberhasilan atau kegagalan memproses suatu peristiwa seringkali lebih penting daripada mengembalikan nilai. Fitur seperti antrian surat mati (DLQ) di Lambda disediakan untuk memastikan Anda dapat mengidentifikasi dan mencoba lagi peristiwa yang gagal, tanpa perlu memberi tahu penelepon.

Debugging di seluruh layanan dan fungsi

Debugging sistem event-driven juga berbeda dibandingkan dengan aplikasi monolitik. Dengan berbagai sistem dan layanan yang melewati peristiwa, tidak mungkin untuk merekam dan

mereproduksi status yang tepat dari beberapa layanan ketika kesalahan terjadi. Karena setiap layanan dan fungsi pemanggilan memiliki file log terpisah, akan lebih rumit untuk menentukan apa yang terjadi pada peristiwa tertentu yang menyebabkan kesalahan.

Ada tiga persyaratan penting untuk membangun pendekatan debugging yang sukses dalam sistem berbasis peristiwa. Pertama, sistem logging yang kuat sangat penting, dan ini disediakan di seluruh AWS layanan dan disematkan dalam fungsi Lambda oleh Amazon. CloudWatch Kedua, dalam sistem ini, penting untuk memastikan bahwa setiap peristiwa memiliki pengenal transaksi yang dicatat pada setiap langkah selama transaksi, untuk membantu saat mencari log.

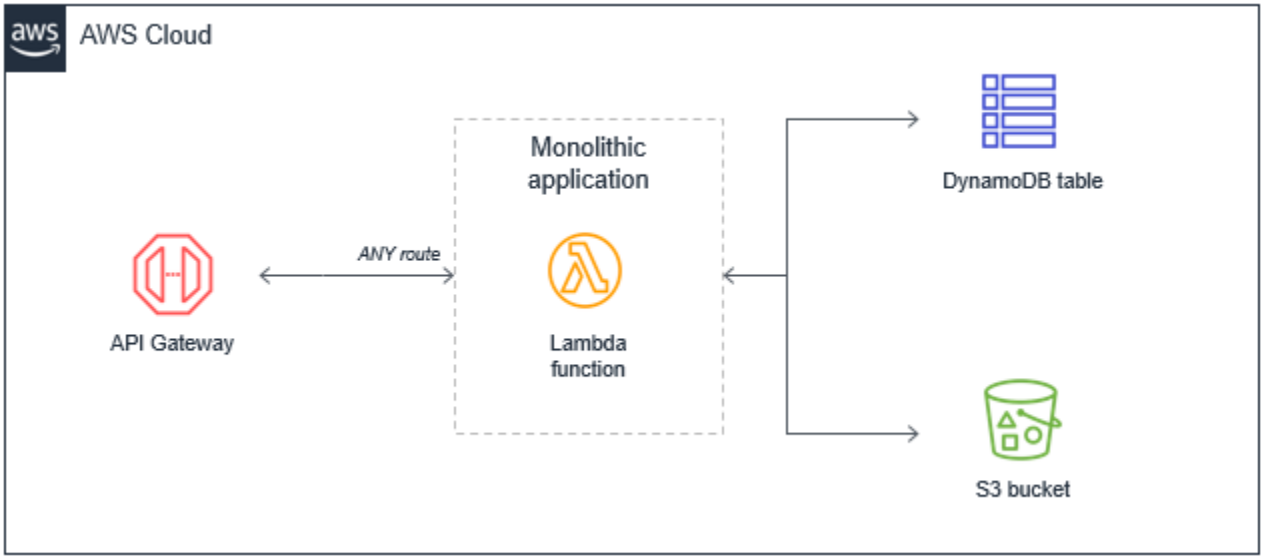
Terakhir, sangat disarankan untuk mengotomatiskan penguraian dan analisis log dengan menggunakan layanan debugging dan pemantauan seperti. AWS X-Ray Ini dapat menggunakan log di beberapa pemanggilan dan layanan Lambda, membuatnya lebih mudah untuk menentukan akar penyebab masalah. Lihat Panduan pemecahan masalah untuk cakupan mendalam tentang penggunaan untuk pemecahan masalah. X-Ray

Anti-patterns dalam Lambda-based aplikasi berbasis peristiwa

Saat membangun arsitektur berbasis peristiwa dengan Lambda, hindari anti-pola umum berikut. Pola-pola ini bekerja tetapi dapat meningkatkan biaya dan kompleksitas.

Monolit Lambda

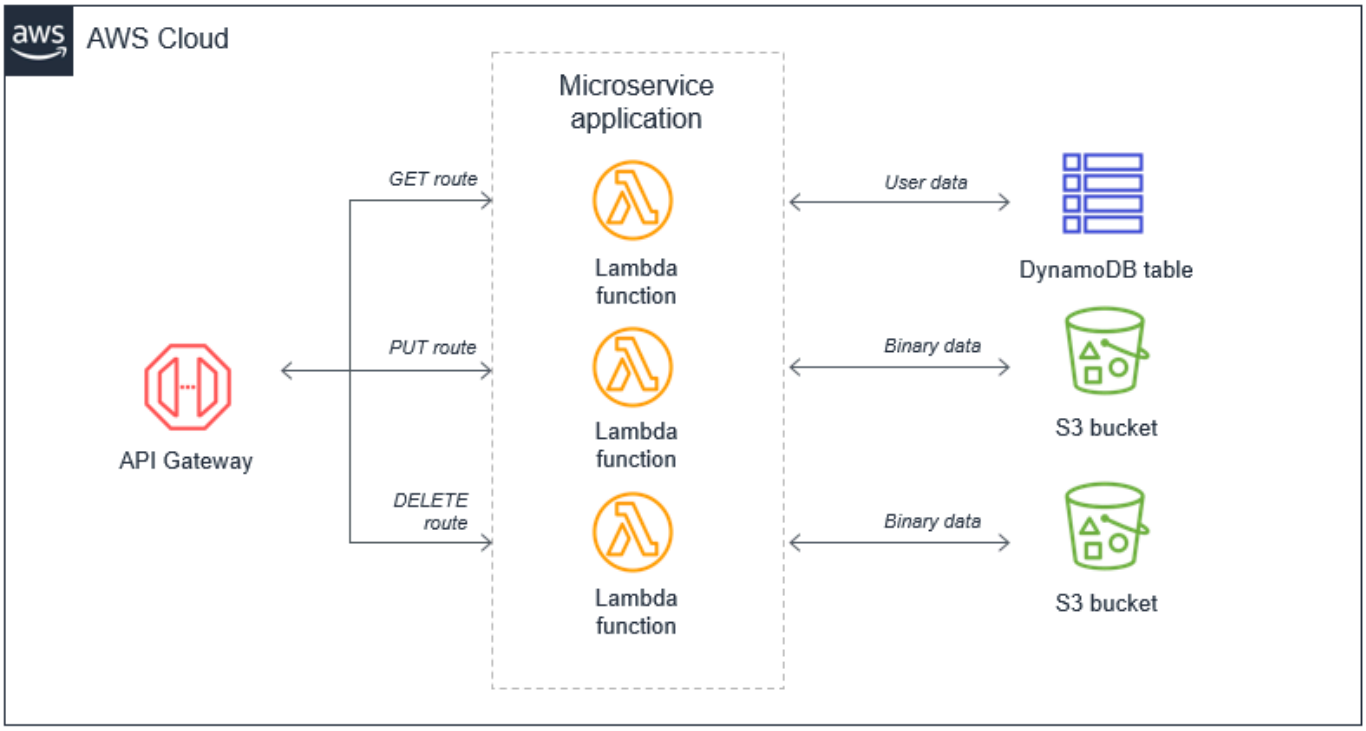
Dalam banyak aplikasi yang dimigrasi dari server tradisional, seperti instans Amazon EC2 atau aplikasi Elastic Beanstalk, pengembang “mengangkat dan menggeser” kode yang ada. Seringkali, ini menghasilkan fungsi Lambda tunggal yang berisi semua logika aplikasi yang dipicu untuk semua peristiwa. Untuk aplikasi web dasar, fungsi Lambda monolitik akan menangani semua rute API Gateway dan terintegrasi dengan semua sumber daya hilir yang diperlukan.



Pendekatan ini memiliki beberapa kelemahan:

- Ukuran paket — Fungsi Lambda mungkin jauh lebih besar karena berisi semua kode yang mungkin untuk semua jalur, yang membuatnya lebih lambat untuk menjalankan layanan Lambda.
- Sulit untuk menerapkan hak istimewa paling sedikit - Peran eksekusi fungsi harus memungkinkan izin ke semua sumber daya yang diperlukan untuk semua jalur, membuat izin menjadi sangat luas. Ini adalah masalah keamanan. Banyak jalur di monolit fungsional tidak memerlukan semua izin yang telah diberikan.
- Lebih sulit untuk ditingkatkan — Dalam sistem produksi, setiap peningkatan ke fungsi tunggal lebih berisiko dan dapat merusak seluruh aplikasi. Memutakhirkan satu jalur dalam fungsi Lambda adalah peningkatan ke seluruh fungsi.
- Lebih sulit untuk mempertahankan - Lebih sulit untuk memiliki beberapa pengembang bekerja pada layanan karena ini adalah repositori kode monolitik. Ini juga meningkatkan beban kognitif pada pengembang dan membuatnya lebih sulit untuk membuat cakupan tes yang sesuai untuk kode.
- Lebih sulit untuk menggunakan kembali kode - Lebih sulit untuk memisahkan pustaka yang dapat digunakan kembali dari monolit, membuat penggunaan kembali kode menjadi lebih sulit. Saat Anda mengembangkan dan mendukung lebih banyak proyek, ini dapat mempersulit untuk mendukung kode dan meningkatkan kecepatan tim Anda.
- Lebih sulit untuk diuji - Ketika baris kode meningkat, menjadi lebih sulit untuk menguji unit semua kemungkinan kombinasi input dan titik masuk dalam basis kode. Umumnya lebih mudah untuk menerapkan pengujian unit untuk layanan yang lebih kecil dengan kode yang lebih sedikit.

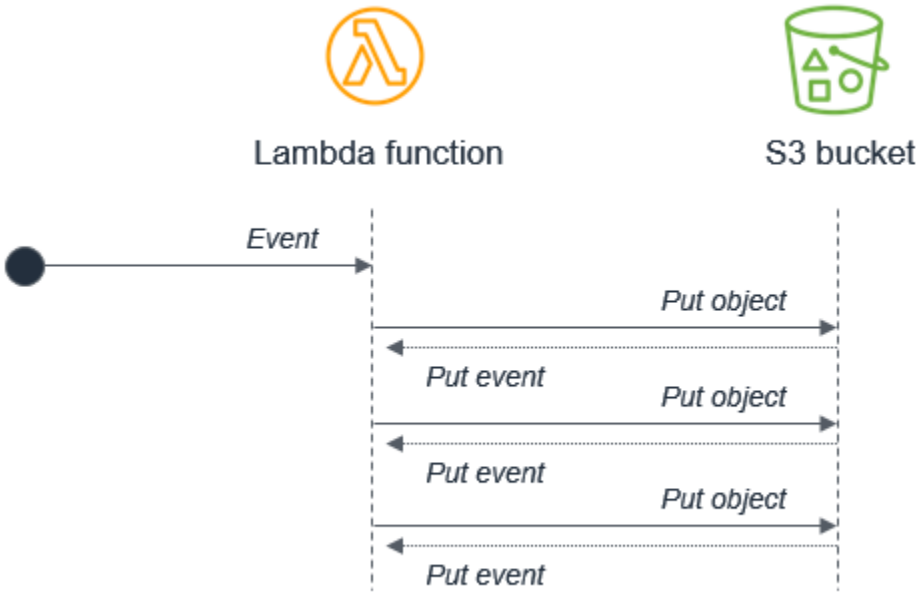
Alternatif yang lebih disukai adalah memecah fungsi Lambda monolitik menjadi layanan mikro individu, memetakan fungsi Lambda tunggal menjadi satu tugas yang terdefinisi dengan baik. Dalam aplikasi web sederhana ini dengan beberapa titik akhir API, arsitektur berbasis layanan mikro yang dihasilkan dapat didasarkan pada rute API Gateway.



Pola rekursif yang menyebabkan fungsi Lambda yang tak terkendali

AWS layanan menghasilkan peristiwa yang memanggil fungsi Lambda, dan fungsi Lambda dapat mengirim pesan ke layanan. AWS Umumnya, layanan atau sumber daya yang memanggil fungsi Lambda harus berbeda dengan layanan atau sumber daya yang dihasilkan fungsi tersebut. Kegagalan untuk mengelola ini dapat mengakibatkan loop tak terbatas.

Misalnya, fungsi Lambda menulis objek ke objek Amazon S3, yang pada gilirannya memanggil fungsi Lambda yang sama dengan menggunakan peristiwa put. Pemanggilan menyebabkan objek kedua ditulis ke bucket, yang memanggil fungsi Lambda yang sama:



Sementara potensi loop tak terbatas ada di sebagian besar bahasa pemrograman, anti-pola ini memiliki potensi untuk mengkonsumsi lebih banyak sumber daya dalam aplikasi tanpa server. Baik Lambda dan Amazon S3 secara otomatis menskalakan berdasarkan lalu lintas, sehingga loop dapat menyebabkan Lambda menskalakan untuk mengkonsumsi semua konkurensi yang tersedia dan Amazon S3 terus menulis objek dan menghasilkan lebih banyak peristiwa untuk Lambda.

Contoh ini menggunakan S3, tetapi risiko loop rekursif juga ada di Amazon SNS, Amazon SQS, DynamoDB, dan layanan lainnya. Anda dapat menggunakan deteksi loop rekursif untuk menemukan dan menghindari anti-pola ini.

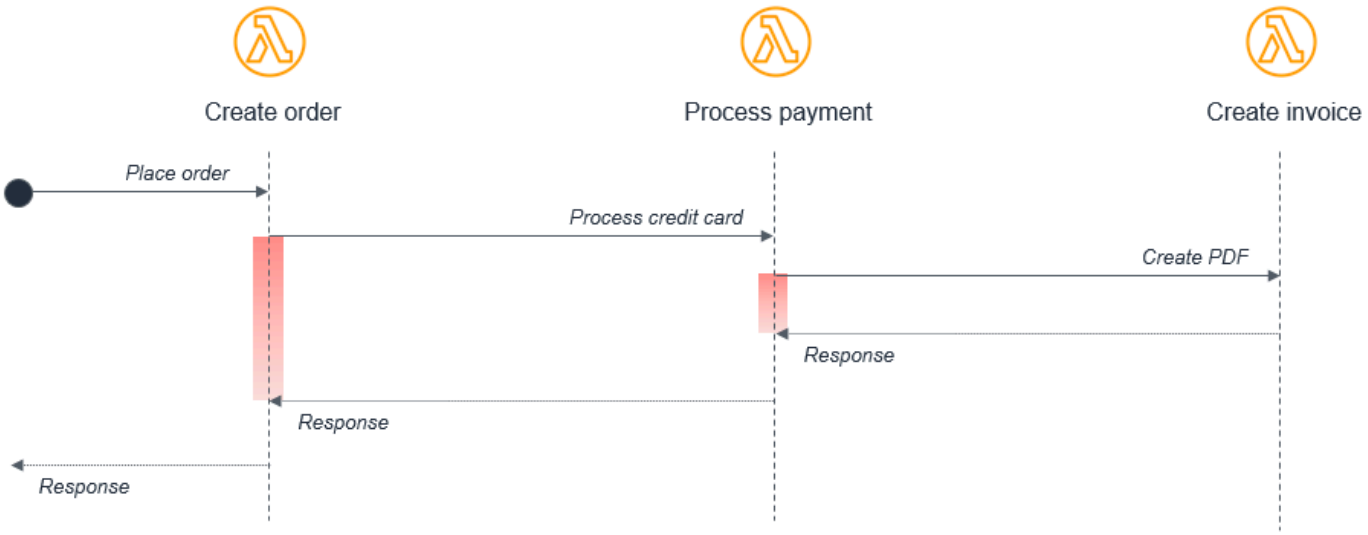
Fungsi Lambda memanggil fungsi Lambda

Fungsi memungkinkan enkapsulasi dan penggunaan kembali kode. Sebagian besar bahasa pemrograman mendukung konsep kode serempak memanggil fungsi dalam basis kode. Dalam hal ini, pemanggil menunggu sampai fungsi mengembalikan respons.

> **Note**
>
> Sementara fungsi Lambda secara langsung memanggil fungsi Lambda lainnya umumnya merupakan anti-pola karena masalah biaya dan kompleksitas, ini tidak berlaku untuk fungsi tahan lama, yang secara khusus dirancang untuk mengatur alur kerja multi-langkah dengan menjalankan fungsi lain.

Ketika ini terjadi pada server tradisional atau instance virtual, penjadwal sistem operasi beralih ke pekerjaan lain yang tersedia. Apakah CPU berjalan pada 0% atau 100% tidak mempengaruhi keseluruhan biaya aplikasi, karena Anda membayar biaya tetap untuk memiliki dan mengoperasikan server.

Model ini sering tidak beradaptasi dengan baik untuk pengembangan tanpa server. Misalnya, pertimbangkan aplikasi e-commerce sederhana yang terdiri dari tiga fungsi Lambda yang memproses pesanan:



Dalam hal ini, fungsi Buat pesanan memanggil fungsi Pembayaran proses, yang pada gilirannya memanggil fungsi Buat faktur. Meskipun aliran sinkron ini mungkin bekerja dalam satu aplikasi di server, ini memperkenalkan beberapa masalah yang dapat dihindari dalam arsitektur tanpa server terdistribusi:

- Biaya — Dengan Lambda, Anda membayar selama doa. Dalam contoh ini, saat fungsi Create invoice berjalan, dua fungsi lainnya juga berjalan dalam keadaan tunggu, ditunjukkan dengan warna merah pada diagram.
- Penanganan kesalahan - Dalam pemanggilan bersarang, penanganan kesalahan bisa menjadi jauh lebih kompleks. Misalnya, kesalahan dalam Create invoice mungkin memerlukan fungsi pembayaran Proses untuk membalikkan tagihan, atau mungkin mencoba lagi proses Create invoice.
- Kopling ketat — Memproses pembayaran biasanya membutuhkan waktu lebih lama daripada membuat faktur. Dalam model ini, ketersediaan seluruh alur kerja dibatasi oleh fungsi paling lambat.

- Penskalaan — Konkurensi ketiga fungsi harus sama. Dalam sistem yang sibuk, ini menggunakan lebih banyak konkurensi daripada yang dibutuhkan.

Dalam aplikasi tanpa server, ada dua pendekatan umum untuk menghindari pola ini. Pertama, gunakan antrean Amazon SQS antara fungsi Lambda. Jika proses hilir lebih lambat daripada proses hulu, antrian tahan lama mempertahankan pesan dan memisahkan dua fungsi. Dalam contoh ini, fungsi Create order akan memublikasikan pesan ke antrean Amazon SQS, dan fungsi pembayaran Proses menggunakan pesan dari antrian.

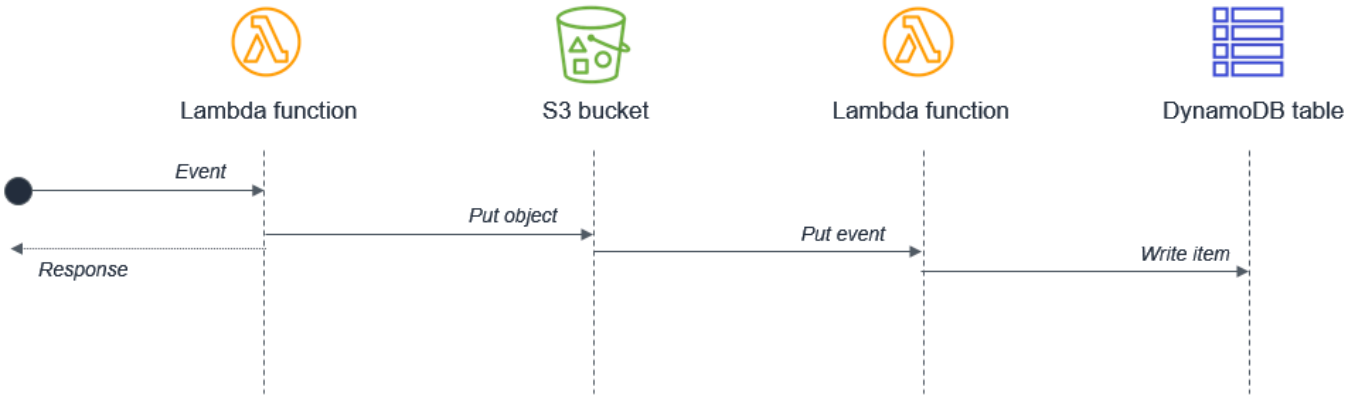
Pendekatan kedua adalah menggunakan AWS Step Functions. Untuk proses kompleks dengan beberapa jenis kegagalan dan logika coba lagi, Step Functions dapat membantu mengurangi jumlah kode kustom yang diperlukan untuk mengatur alur kerja. Akibatnya, Step Functions mengatur pekerjaan dan menangani kesalahan dan percobaan ulang dengan kuat, dan fungsi Lambda hanya berisi logika bisnis.

Penantian sinkron dalam satu fungsi Lambda

Pastikan bahwa aktivitas yang berpotensi bersamaan tidak dijadwalkan secara sinkron dalam satu fungsi Lambda. Misalnya, fungsi Lambda mungkin menulis ke bucket S3 dan kemudian menulis ke tabel DynamoDB:



Dalam desain ini, waktu tunggu diperparah karena aktivitasnya berurutan. Dalam kasus di mana tugas kedua tergantung pada penyelesaian tugas pertama, Anda dapat mengurangi total waktu tunggu dan biaya eksekusi dengan memiliki dua fungsi Lambda terpisah:



Dalam desain ini, fungsi Lambda pertama merespons segera setelah meletakkan objek ke bucket Amazon S3. Layanan S3 memanggil fungsi Lambda kedua, yang kemudian menulis data ke tabel DynamoDB. Pendekatan ini meminimalkan total waktu tunggu dalam eksekusi fungsi Lambda.

Merancang aplikasi Lambda

Aplikasi berbasis peristiwa yang dirancang dengan baik menggunakan kombinasi AWS layanan dan kode khusus untuk memproses dan mengelola permintaan dan data. Bab ini berfokus pada Lambda-specific topik dalam desain aplikasi. Ada banyak pertimbangan penting bagi arsitek tanpa server saat merancang aplikasi untuk sistem produksi yang sibuk.

Banyak praktik terbaik yang berlaku untuk pengembangan perangkat lunak dan sistem terdistribusi juga berlaku untuk pengembangan aplikasi tanpa server. Tujuan keseluruhannya adalah untuk mengembangkan beban kerja yaitu:

- Andal - menawarkan pengguna akhir Anda tingkat ketersediaan yang tinggi. AWS layanan tanpa server dapat diandalkan karena mereka juga dirancang untuk kegagalan.
- Tahan lama — menyediakan opsi penyimpanan yang memenuhi kebutuhan daya tahan beban kerja Anda.
- Aman — mengikuti praktik terbaik dan menggunakan alat yang disediakan untuk mengamankan akses ke beban kerja dan membatasi radius ledakan.
- Performant — menggunakan sumber daya komputasi secara efisien dan memenuhi kebutuhan kinerja pengguna akhir Anda.
- Cost-efficient— merancang arsitektur yang menghindari biaya yang tidak perlu yang dapat diskalakan tanpa pengeluaran berlebihan, dan juga dinonaktifkan tanpa overhead yang signifikan.

Prinsip-prinsip desain berikut dapat membantu Anda membangun beban kerja yang memenuhi tujuan ini. Tidak setiap prinsip mungkin berlaku untuk setiap arsitektur, tetapi mereka harus memandu Anda dalam keputusan arsitektur umum.

Topik

- Gunakan layanan alih-alih kode khusus
- Memahami tingkat abstraksi Lambda
- Menerapkan tanpa kewarganegaraan dalam fungsi
- Minimalkan kopling
- Bangun untuk data sesuai permintaan, bukan batch
- Pilih opsi orkestrasi untuk alur kerja yang kompleks
- Menerapkan idempotensi
- Gunakan beberapa AWS akun untuk mengelola kuota

Gunakan layanan alih-alih kode khusus

Aplikasi tanpa server biasanya terdiri dari beberapa AWS layanan, terintegrasi dengan kode khusus yang dijalankan dalam fungsi Lambda. Sementara Lambda dapat diintegrasikan dengan sebagian besar AWS layanan, layanan yang paling umum digunakan dalam aplikasi tanpa server adalah:

Kategori	AWS layanan
Hitung	AWS Lambda
Penyimpanan data	Amazon S3 Amazon DynamoDB Amazon RDS
API	Amazon API Gateway
Integrasi aplikasi	Amazon EventBridge Amazon SNS Amazon SQS
Orkestrasi	Lambda fungsi tahan lama AWS Step Functions
Streaming data dan analitik	Amazon Data Firehose

> **Note**
>
> Banyak layanan tanpa server menyediakan replikasi dan dukungan untuk beberapa Wilayah, termasuk DynamoDB dan Amazon S3. Fungsi Lambda dapat digunakan di beberapa Wilayah sebagai bagian dari pipeline penerapan, dan API Gateway dapat dikonfigurasi untuk mendukung konfigurasi ini. Lihat contoh arsitektur ini yang menunjukkan bagaimana hal ini dapat dicapai.

Ada banyak pola umum yang mapan dalam arsitektur terdistribusi yang dapat Anda bangun sendiri atau terapkan menggunakan AWS layanan. Bagi sebagian besar pelanggan, ada sedikit nilai komersial dalam menginvestasikan waktu untuk mengembangkan pola-pola ini dari awal. Ketika aplikasi Anda membutuhkan salah satu pola ini, gunakan AWS layanan yang sesuai:

Pola	AWS layanan
Antrean	Amazon SQS
Bus acara	Amazon EventBridge
Publish/subscribe (kipas)	Amazon SNS
Orkestrasi	Lambda fungsi tahan lama AWS Step Functions
API	Amazon API Gateway
Aliran acara	Amazon Kinesis

Layanan ini dirancang untuk berintegrasi dengan Lambda dan Anda dapat menggunakan infrastruktur sebagai kode (IAc) untuk membuat dan membuang sumber daya dalam layanan. Anda dapat menggunakan salah satu layanan ini melalui AWS SDK tanpa perlu menginstal aplikasi atau mengkonfigurasi server. Menjadi mahir menggunakan layanan ini melalui kode dalam fungsi Lambda Anda adalah langkah penting untuk menghasilkan aplikasi tanpa server yang dirancang dengan baik.

Memahami tingkat abstraksi Lambda

Layanan Lambda membatasi akses Anda ke sistem operasi, hypervisor, dan perangkat keras yang mendasari yang menjalankan fungsi Lambda Anda. Layanan terus meningkatkan dan mengubah infrastruktur untuk menambah fitur, mengurangi biaya, dan membuat layanan lebih berkinerja. Kode Anda seharusnya tidak mengasumsikan pengetahuan tentang bagaimana Lambda dirancang dan menganggap tidak ada afinitas perangkat keras.

Demikian pula, integrasi Lambda dengan layanan lain dikelola oleh AWS, dengan hanya sejumlah kecil opsi konfigurasi yang terpapar kepada Anda. Misalnya, ketika API Gateway dan Lambda berinteraksi, tidak ada konsep load balancing karena sepenuhnya dikelola oleh layanan. Anda juga tidak memiliki kontrol langsung atas Availability Zone yang digunakan layanan saat menjalankan

fungsi kapan saja, atau bagaimana Lambda menentukan kapan harus meningkatkan atau menurunkan jumlah lingkungan eksekusi.

Abstraksi ini membantu Anda fokus pada aspek integrasi aplikasi Anda, aliran data, dan logika bisnis di mana beban kerja Anda memberikan nilai bagi pengguna akhir Anda. Memungkinkan layanan untuk mengelola mekanisme yang mendasarinya membantu Anda mengembangkan aplikasi lebih cepat dengan lebih sedikit kode khusus untuk dipelihara.

Menerapkan tanpa kewarganegaraan dalam fungsi

Untuk fungsi Lambda, Anda harus berasumsi bahwa lingkungan hanya ada untuk satu pemanggilan. Fungsi harus menginisialisasi status yang diperlukan saat pertama kali dimulai. Misalnya, fungsi Anda mungkin memerlukan pengambilan data dari tabel DynamoDB. Ini harus melakukan perubahan data permanen ke toko yang tahan lama seperti Amazon S3, DynamoDB, atau Amazon SQS sebelum keluar. Seharusnya tidak bergantung pada struktur data yang ada atau file sementara, atau keadaan internal apa pun yang akan dikelola oleh beberapa pemanggilan.

Untuk menginisialisasi koneksi database dan pustaka, atau status beban, Anda dapat memanfaatkan inisialisasi statis. Karena lingkungan eksekusi digunakan kembali jika memungkinkan untuk meningkatkan kinerja, Anda dapat mengamortisasi waktu yang dibutuhkan untuk menginisialisasi sumber daya ini melalui beberapa pemanggilan. Namun, Anda tidak boleh menyimpan variabel atau data apa pun yang digunakan dalam fungsi dalam lingkup global ini.

Minimalkan kopling

Sebagian besar arsitektur harus lebih memilih banyak fungsi yang lebih pendek daripada yang lebih sedikit, yang lebih besar. Tujuan dari setiap fungsi harus untuk menangani peristiwa yang diteruskan ke dalam fungsi, tanpa pengetahuan atau harapan dari keseluruhan alur kerja atau volume transaksi. Hal ini membuat fungsi agnostik ke sumber acara dengan kopling minimal ke layanan lain.

Setiap konstanta lingkup global yang jarang berubah harus diimplementasikan sebagai variabel lingkungan untuk memungkinkan pembaruan tanpa penerapan. Setiap rahasia atau informasi sensitif harus disimpan di AWS Systems Manager Parameter Store atau AWS Secrets Manager dan dimuat oleh fungsi. Karena sumber daya ini khusus akun, Anda dapat membuat pipeline build di beberapa akun. Pipeline memuat rahasia yang sesuai per lingkungan, tanpa mengeksposnya ke pengembang atau memerlukan perubahan kode apa pun.

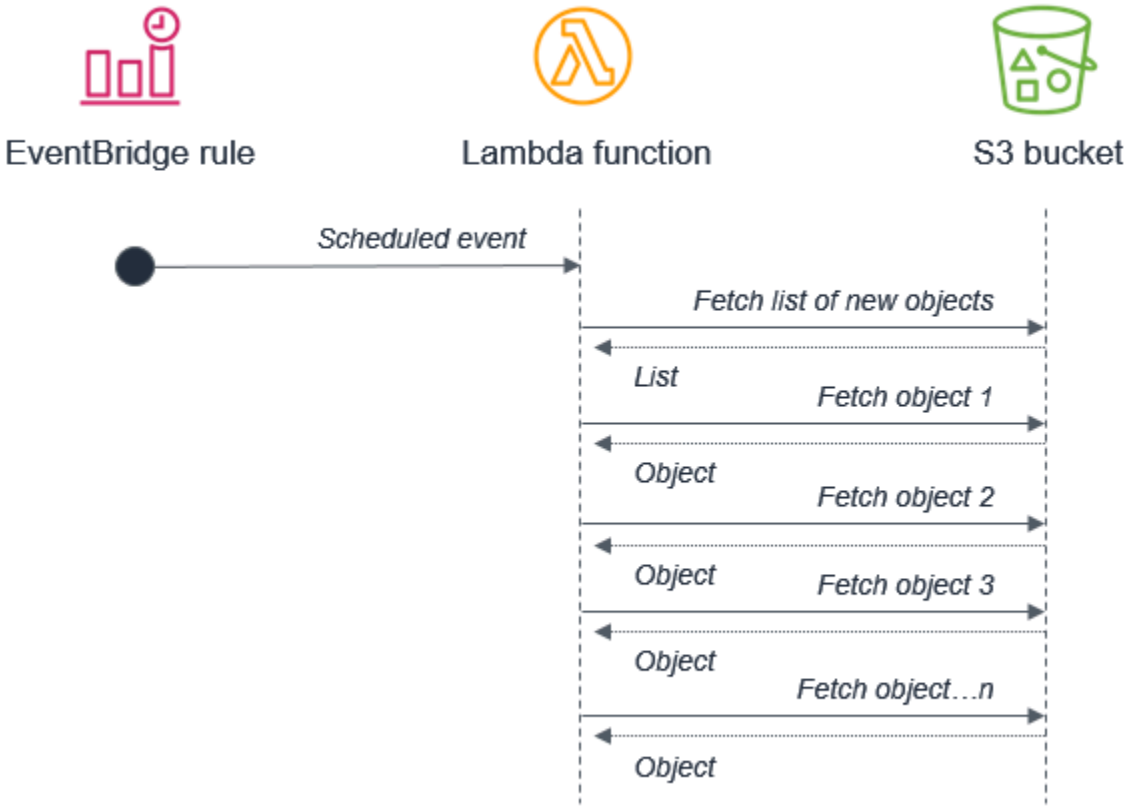
Bangun untuk data sesuai permintaan, bukan batch

Banyak sistem tradisional dirancang untuk berjalan secara berkala dan memproses batch transaksi yang telah dibangun dari waktu ke waktu. Misalnya, aplikasi perbankan mungkin berjalan setiap jam untuk memproses transaksi ATM menjadi buku besar pusat. Dalam Lambda-based aplikasi, pemrosesan kustom harus dipicu oleh setiap peristiwa, memungkinkan layanan untuk meningkatkan konkurensi sesuai kebutuhan, untuk menyediakan pemrosesan transaksi yang hampir real time.

Secara default, pemanggilan fungsi dapat berjalan hingga 15 menit. Pada Instans Terkelola Lambda, pemanggilan asinkron dan pemetaan sumber peristiwa (ESM) dapat berjalan hingga 90 menit. Untuk pemrosesan yang berjalan lebih lama, pos pemeriksaan fungsi tahan lama berkembang di beberapa pemanggilan. Ketika dipanggil secara asinkron, eksekusi yang tahan lama dapat berjalan hingga satu tahun. Namun, Anda tetap harus lebih memilih pemrosesan berbasis peristiwa daripada pemrosesan batch jika memungkinkan.

Meskipun Anda dapat menjalankan tugas cron dalam aplikasi tanpa server dengan menggunakan ekspresi terjadwal untuk aturan di Amazon EventBridge, ini harus digunakan dengan hemat atau sebagai pilihan terakhir. Dalam setiap tugas terjadwal yang memproses batch, ada potensi volume transaksi tumbuh melampaui apa yang dapat diproses dalam batas durasi Lambda. Jika keterbatasan sistem eksternal memaksa Anda untuk menggunakan penjadwal, Anda biasanya harus menjadwalkan untuk periode waktu berulang yang masuk akal terpendek.

Misalnya, bukan praktik terbaik untuk menggunakan proses batch yang memicu fungsi Lambda untuk mengambil daftar objek Amazon S3 baru. Ini karena layanan mungkin menerima lebih banyak objek baru di antara batch daripada yang dapat diproses dalam batas durasi Lambda.



Sebagai gantinya, Amazon S3 harus menjalankan fungsi Lambda setiap kali objek baru dimasukkan ke dalam ember. Pendekatan ini secara signifikan lebih terukur dan bekerja dalam waktu nyaris nyata.



Pilih opsi orkestrasi untuk alur kerja yang kompleks

Alur kerja yang melibatkan logika percabangan, berbagai jenis model kegagalan, dan logika coba lagi biasanya menggunakan orkestrator untuk melacak status eksekusi secara keseluruhan. Jangan membangun orkestrasi ad-hoc dalam fungsi Lambda. Ini menghasilkan kopling yang ketat, kode perutean yang kompleks, dan tidak ada pemulihan status otomatis.

Sebagai gantinya, gunakan salah satu opsi orkestrasi yang dibuat khusus ini:

- Fungsi tahan lama Lambda: Application-centric orkestrasi menggunakan bahasa pemrograman standar dengan pos pemeriksaan otomatis, coba ulang bawaan, dan pemulihan eksekusi. Ideal untuk pengembang yang lebih suka menyimpan logika alur kerja dalam kode bersama logika bisnis dalam Lambda.
- AWS Step Functions: Orkestrasi alur kerja visual dengan integrasi asli ke 220+ layanan. AWS Ideal untuk koordinasi multi-layanan, infrastruktur tanpa perawatan, dan desain alur kerja visual.

Untuk panduan memilih di antara opsi-opsi ini, lihat Fungsi tahan lama atau Step Functions.

Dengan Step Functions, Anda menggunakan mesin status untuk mengelola orkestrasi. Ini mengekstrak penanganan kesalahan, perutean, dan logika percabangan dari kode Anda, menggantinya dengan mesin status yang dideklarasikan menggunakan JSON. Selain membuat alur kerja lebih kuat dan dapat diamati, Anda juga dapat menambahkan versi ke alur kerja dan menjadikan mesin status sebagai sumber daya terkodifikasi yang dapat Anda tambahkan ke repositori kode.

Adalah umum untuk alur kerja yang lebih sederhana dalam fungsi Lambda menjadi lebih kompleks dari waktu ke waktu. Saat mengoperasikan aplikasi tanpa server produksi, penting untuk mengidentifikasi kapan ini terjadi, sehingga Anda dapat memigrasikan logika ini ke mesin status atau fungsi tahan lama.

Menerapkan idempotensi

AWS Layanan tanpa server, termasuk Lambda, toleran terhadap kesalahan dan dirancang untuk menangani kegagalan. Misalnya, jika layanan memanggil fungsi Lambda dan ada gangguan layanan, Lambda memanggil fungsi Anda di Availability Zone yang berbeda. Jika fungsi Anda memunculkan kesalahan, Lambda mencoba ulang pemanggilan.

Karena acara yang sama mungkin diterima lebih dari sekali, fungsi harus dirancang untuk menjadi idempoten. Ini berarti bahwa menerima acara yang sama beberapa kali tidak mengubah hasil di luar pertama kali acara diterima.

Anda dapat menerapkan idempotensi dalam fungsi Lambda dengan menggunakan tabel DynamoDB untuk melacak pengidentifikasi yang baru diproses untuk menentukan apakah transaksi telah ditangani sebelumnya. Tabel DynamoDB biasanya mengimplementasikan nilai Time To Live (TTL) untuk item kedaluwarsa untuk membatasi ruang penyimpanan yang digunakan.

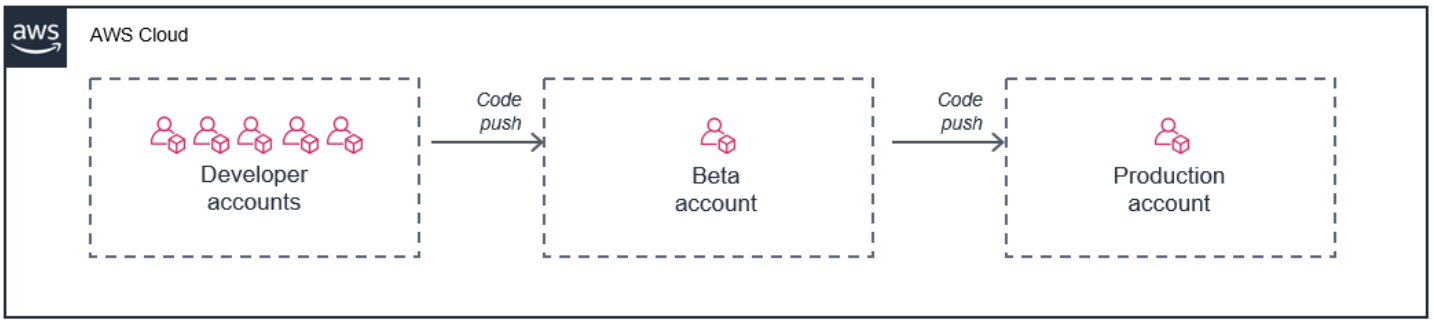
Gunakan beberapa AWS akun untuk mengelola kuota

Banyak kuota layanan di AWS ditetapkan pada tingkat akun. Ini berarti bahwa saat Anda menambahkan lebih banyak beban kerja, Anda dapat dengan cepat menghabiskan batas Anda.

Cara efektif untuk mengatasi masalah ini adalah dengan menggunakan banyak AWS akun, mendedikasikan setiap beban kerja ke akunnya sendiri. Ini mencegah kuota dibagikan dengan beban kerja lain atau sumber daya non-produksi.

Selain itu, dengan menggunakan AWS Organizations, Anda dapat mengelola penagihan, kepatuhan, dan keamanan akun ini secara terpusat. Anda dapat melampirkan kebijakan ke grup akun untuk menghindari skrip kustom dan proses manual.

Salah satu pendekatan umum adalah menyediakan setiap pengembang dengan AWS akun, dan kemudian menggunakan akun terpisah untuk tahap penerapan beta dan produksi:



Dalam model ini, setiap pengembang memiliki batasan sendiri untuk akun, sehingga penggunaannya tidak memengaruhi lingkungan produksi Anda. Pendekatan ini juga memungkinkan pengembang untuk menguji fungsi Lambda secara lokal pada mesin pengembangan mereka terhadap sumber daya cloud langsung di akun masing-masing.

Buat fungsi Lambda pertama Anda

Untuk memulai Lambda, gunakan konsol Lambda untuk membuat fungsi. Dalam beberapa menit, Anda dapat membuat dan menerapkan fungsi dan mengujinya di konsol.

Saat Anda melakukan tutorial, Anda akan mempelajari beberapa konsep dasar Lambda, seperti cara meneruskan argumen ke fungsi Anda menggunakan Lambda objek acara. Anda juga akan mempelajari cara mengembalikan output log dari fungsi Anda, dan cara melihat log pemanggilan fungsi Anda di Amazon CloudWatch Log.

Agar semuanya tetap sederhana, Anda membuat fungsi Anda menggunakan Python atau Node.js waktu pelaksanaan. Dengan bahasa yang ditafsirkan ini, Anda dapat mengedit kode fungsi langsung di editor kode bawaan konsol. Dengan bahasa yang dikompilasi seperti Java dan C#, Anda harus membuat paket penerapan pada mesin build lokal Anda dan mengunggahnya ke Lambda. Untuk mempelajari tentang menerapkan fungsi ke Lambda menggunakan runtime lain, lihat tautan di the section called “Langkah selanjutnya” bagian.

> **Tip**
>
> Untuk mempelajari cara membangun solusi tanpa server, lihat Panduan Pengembang Tanpa Server.

Mendaftar untuk Akun AWS

Untuk memulai dengan AWS, Anda membutuhkan Akun AWS. Ada dua cara untuk mendaftar AWS. Jika Anda ingin kontrol penuh atas konfigurasi akun dan tempat penyimpanan konten Anda, atau jika Anda telah mengatur beban kerja seperti HIPAA atau FedRAMP, gunakan Mendaftar untuk AWS (lanjutan). Jika Anda adalah pengguna baru dan ingin menggunakan login yang sudah Anda miliki untuk masuk AWS dan buat AWS lingkungan dengan default yang telah dikonfigurasi sebelumnya dalam pilihan Wilayah AWS, gunakan Mendaftar untuk AWS.

Membuat fungsi Lambda dengan konsol

Dalam contoh ini, fungsi Anda mengambil objek JSON yang berisi dua nilai integer berlabel `"length"` dan `"width"`. Fungsi mengalikan nilai-nilai ini untuk menghitung area dan mengembalikannya sebagai string JSON.

Fungsi Anda juga mencetak area yang dihitung, bersama dengan namanya CloudWatch grup log. Kemudian dalam tutorial, Anda akan belajar menggunakan CloudWatch Log untuk melihat catatan pemanggilan fungsi Anda.

Untuk membuat fungsi Hello world Lambda dengan konsol

1. Buka Halaman Fungsi Acara dari konsol Lambda.
2. Memilih Buat Fungsi Acara.
3. Pilih Penulis dari awal.
4. Di Informasi dasar panel, untuk Nama fungsi, masukkan **`myLambdaFunction`**.
5. Untuk Waktu pelaksanaan, pilih salah satu Node.js 24 atau Python 3.14.
6. Tinggalkan arsitektur setel ke x86_64, dan kemudian memilih Buat Fungsi Acara.

Selain fungsi sederhana yang mengembalikan pesan `Hello from Lambda!`, Lambda juga menciptakan peran eksekusi untuk fungsi Anda. Peran eksekusi adalah AWS Identity and Access Management Peran (IAM) yang memberikan izin fungsi Lambda untuk mengakses Layanan AWS dan sumber daya. Untuk fungsi Anda, peran yang dibuat Lambda memberikan izin dasar untuk menulis CloudWatch Log.

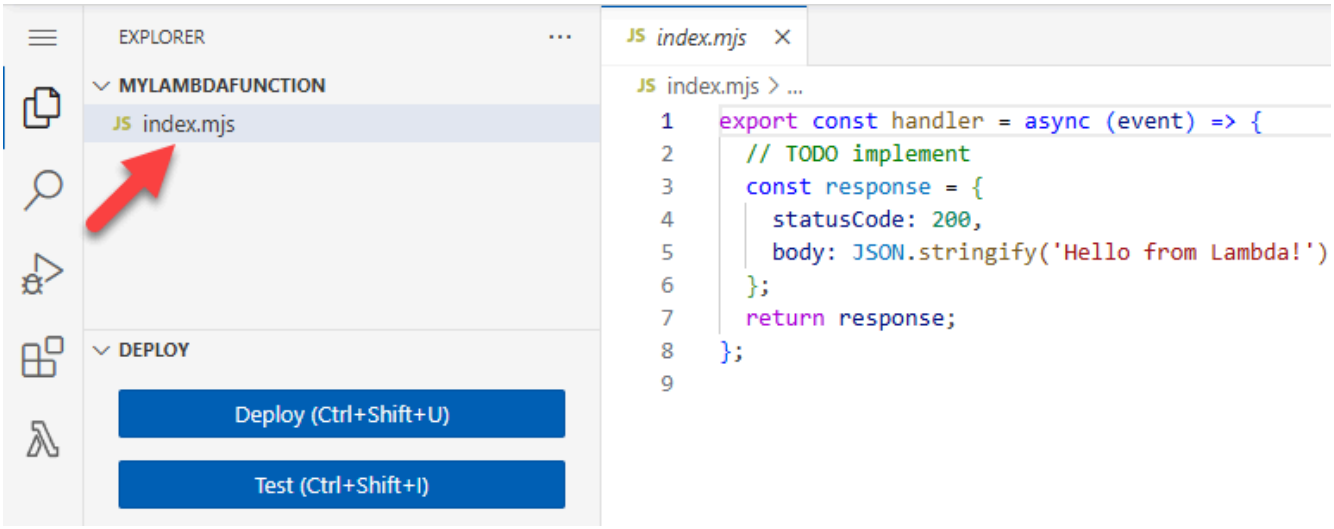
Gunakan editor kode bawaan konsol untuk mengganti kode Hello world yang dibuat Lambda dengan kode fungsi Anda sendiri.

Node.js

Untuk memodifikasi kode di konsol

1. Pilih Kode tab.

   Di editor kode bawaan konsol, Anda akan melihat kode fungsi yang dibuat Lambda. Jika Anda tidak melihat index.mjs tab di editor kode, pilih index.mjs di file explorer seperti yang ditunjukkan pada diagram berikut.



2. Tempelkan kode berikut ke index.mjs tab, menggantikan kode yang dibuat Lambda.

```
export const handler = async (event, context) => {

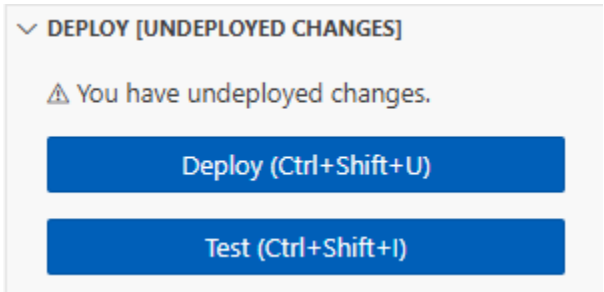
  const length = event.length;
  const width = event.width;
  let area = calculateArea(length, width);
  console.log(`The area is ${area}`);

  console.log('CloudWatch log group: ', context.logGroupName);

  let data = {
    "area": area,
  };
    return JSON.stringify(data);

  function calculateArea(length, width) {
    return length * width;
  }
};
```

3. Di MENYEBARKAN bagian, pilih Menyebarkan untuk memperbarui kode fungsi Anda:



Memahami kode fungsi Anda

Sebelum Anda pindah ke langkah berikutnya, mari kita luangkan waktu sejenak untuk melihat kode fungsi dan memahami beberapa konsep Lambda kunci.

- Handler Lambda:

  Fungsi Lambda Anda berisi Node.js fungsi bernama `handler`. Fungsi Lambda di Node.js dapat berisi lebih dari satu Node.js fungsinya, tetapi pengurus fungsi selalu menjadi titik masuk ke kode Anda. Ketika fungsi Anda dipanggil, Lambda menjalankan metode ini.

  Saat Anda membuat fungsi Hello world menggunakan konsol, Lambda secara otomatis mengatur nama metode handler untuk fungsi Anda ke `handler`. Pastikan untuk tidak mengedit nama ini Node.js fungsi. Jika Anda melakukannya, Lambda tidak akan dapat menjalankan kode Anda saat Anda memanggil fungsi Anda.

  Untuk mempelajari selengkapnya tentang handler Lambda di Node.js, lihat the section called “Handler”.
- Objek peristiwa Lambda:

  Fungsi `handler` membutuhkan dua argumen, `event` dan `context`. Sebuah peristiwa di Lambda adalah dokumen berformat JSON yang berisi data untuk diproses fungsi Anda.

  Jika fungsi Anda dipanggil oleh orang lain Layanan AWS, objek acara berisi informasi tentang peristiwa yang menyebabkan pemanggilan. Misalnya, jika fungsi Anda dipanggil saat objek diunggah ke bucket Amazon Simple Storage Service (Amazon S3), acara tersebut berisi nama bucket dan kunci objek.

  Dalam contoh ini, Anda akan membuat acara di konsol dengan memasukkan dokumen berformat JSON dengan dua pasangan kunci-nilai.
- Objek konteks Lambda:

  Argumen kedua yang diambil fungsi Anda adalah `context`. Lambda melewati objek konteks untuk fungsi Anda secara otomatis. Objek konteks berisi informasi tentang pemanggilan fungsi dan lingkungan eksekusi.

  Anda dapat menggunakan objek konteks untuk menampilkan informasi tentang pemanggilan fungsi Anda untuk tujuan pemantauan. Dalam contoh ini, fungsi Anda menggunakan `logGroupName` parameter untuk menampilkan nama nya CloudWatch grup log.

Untuk mempelajari lebih lanjut tentang objek konteks Lambda di Node.js, lihat the section called "Konteks".

- Masuk ke Lambda:

  Dengan Node.js, Anda dapat menggunakan metode konsol seperti `console.log` dan `console.error` untuk mengirim informasi ke log fungsi Anda. Contoh kode menggunakan `console.log` pernyataan untuk menampilkan area yang dihitung dan nama fungsi CloudWatch Grup log. Anda juga dapat menggunakan pustaka logging apa pun yang menulis ke `stdout` atau `stderr`.

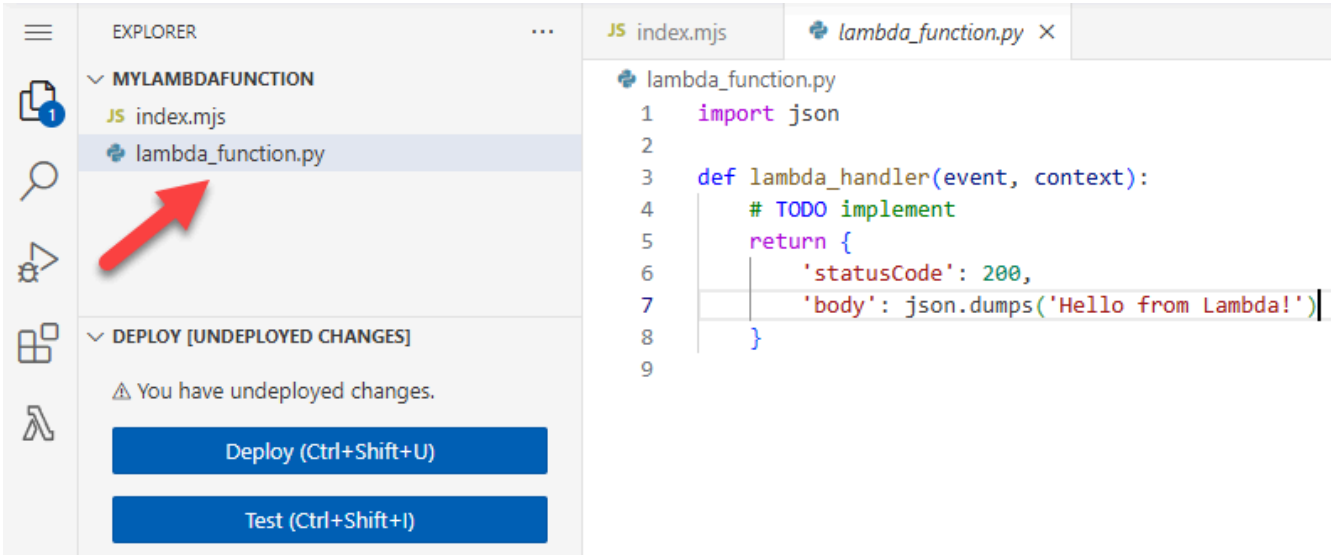
  Untuk mempelajari selengkapnya, lihat the section called "Pencatatan log". Untuk mempelajari cara masuk dalam runtime lain, lihat halaman 'Membangun dengan' untuk runtime yang Anda minati.

Python

Untuk memodifikasi kode di konsol

1. Pilih Kode tab.

   Di editor kode bawaan konsol, Anda akan melihat kode fungsi yang dibuat Lambda. Jika Anda tidak melihat lambda_function.py tab di editor kode, pilih lambda_function.py di file explorer seperti yang ditunjukkan pada diagram berikut.



2. Tempelkan kode berikut ke lambda_function.py tab, menggantikan kode yang dibuat Lambda.

```
import json
import logging

logger = logging.getLogger()
logger.setLevel(logging.INFO)

def lambda_handler(event, context):

    # Get the length and width parameters from the event object. The
    # runtime converts the event object to a Python dictionary
    length = event['length']
    width = event['width']

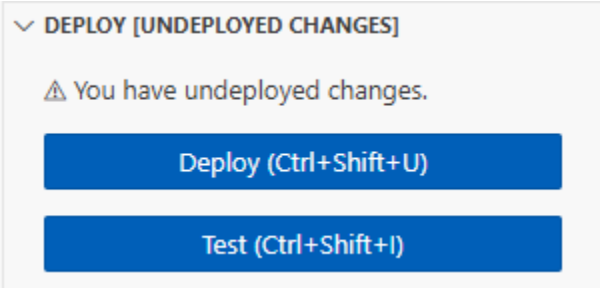
    area = calculate_area(length, width)
    print(f"The area is {area}")

    logger.info(f"CloudWatch logs group: {context.log_group_name}")

    # return the calculated area as a JSON string
    data = {"area": area}
    return json.dumps(data)

def calculate_area(length, width):
    return length*width
```

3. Di MENYEBARKAN bagian, pilih Menyebarkan untuk memperbarui kode fungsi Anda:



Memahami kode fungsi Anda

Sebelum Anda pindah ke langkah berikutnya, mari kita luangkan waktu sejenak untuk melihat kode fungsi dan memahami beberapa konsep Lambda kunci.

- Handler Lambda:

Fungsi Lambda Anda berisi fungsi Python bernama `lambda_handler`. Fungsi Lambda dalam Python dapat berisi lebih dari satu fungsi Python, tetapi pengurus fungsi selalu menjadi titik masuk ke kode Anda. Ketika fungsi Anda dipanggil, Lambda menjalankan metode ini.

Saat Anda membuat fungsi Hello world menggunakan konsol, Lambda secara otomatis mengatur nama metode handler untuk fungsi Anda ke `lambda_handler`. Pastikan untuk tidak mengedit nama fungsi Python ini. Jika Anda melakukannya, Lambda tidak akan dapat menjalankan kode Anda saat Anda memanggil fungsi Anda.

Untuk mempelajari selengkapnya tentang handler Lambda di Python, lihat the section called "Handler".

- Objek peristiwa Lambda:

  Fungsi `lambda_handler` membutuhkan dua argumen, `event` dan `context`. Sebuah peristiwa di Lambda adalah dokumen berformat JSON yang berisi data untuk diproses fungsi Anda.

  Jika fungsi Anda dipanggil oleh orang lain Layanan AWS, objek acara berisi informasi tentang peristiwa yang menyebabkan pemanggilan. Misalnya, jika fungsi Anda dipanggil saat objek diunggah ke bucket Amazon Simple Storage Service (Amazon S3), acara tersebut berisi nama bucket dan kunci objek.

  Dalam contoh ini, Anda akan membuat acara di konsol dengan memasukkan dokumen berformat JSON dengan dua pasangan kunci-nilai.

- Objek konteks Lambda:

  Argumen kedua yang diambil fungsi Anda adalah `context`. Lambda melewati objek konteks untuk fungsi Anda secara otomatis. Objek konteks berisi informasi tentang pemanggilan fungsi dan lingkungan eksekusi.

  Anda dapat menggunakan objek konteks untuk menampilkan informasi tentang pemanggilan fungsi Anda untuk tujuan pemantauan. Dalam contoh ini, fungsi Anda menggunakan `log_group_name` parameter untuk menampilkan nama nya CloudWatch grup log.

  Untuk mempelajari lebih lanjut tentang objek konteks Lambda di Python, lihat the section called "Konteks".

- Masuk ke Lambda:

Dengan Python, Anda dapat menggunakan salah satu `print` pernyataan atau pustaka logging Python untuk mengirim informasi ke log fungsi Anda. Untuk mengilustrasikan perbedaan dalam apa yang ditangkap, kode contoh menggunakan kedua metode. Dalam aplikasi produksi, sebaiknya gunakan pustaka logging.

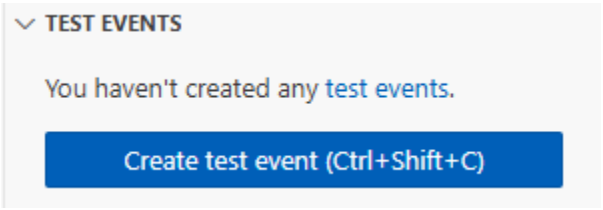
Untuk mempelajari selengkapnya, lihat the section called “Pencatatan log”. Untuk mempelajari cara masuk dalam runtime lain, lihat halaman 'Membangun dengan' untuk runtime yang Anda minati.

Memanggil fungsi Lambda menggunakan editor kode konsol

Untuk memanggil fungsi Anda menggunakan editor kode konsol Lambda, buat acara pengujian untuk dikirim ke fungsi Anda. Acara ini adalah dokumen berformat JSON yang berisi dua pasangan kunci-nilai dengan kunci `"length"` dan `"width"`.

Untuk membuat acara pengujian

1. Di ACARA UJI bagian dari editor kode konsol, pilih Buat acara pengujian.



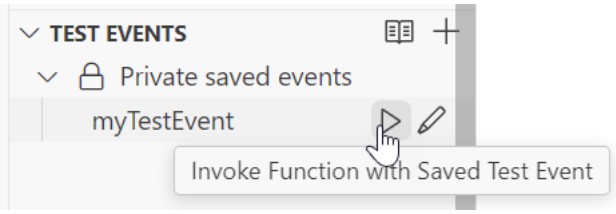
2. Untuk Nama Acara, masukkan **`myTestEvent`**.
3. Di JSON acara bagian, ganti JSON default dengan yang berikut:

```
{
  "length": 6,
  "width": 7
}
```

4. Pilih Simpan.

Untuk menguji fungsi Anda dan melihat catatan pemanggilan

Di ACARA UJI bagian editor kode konsol, pilih ikon jalankan di sebelah acara pengujian Anda:



Ketika fungsi Anda selesai berjalan, log respons dan fungsi ditampilkan di KELUARAN tab. Anda akan melihat hasil yang serupa dengan yang berikut:

Node.js

```
Status: Succeeded
Test Event Name: myTestEvent

Response
"{\"area\":42}"

Function Logs
START RequestId: 5c012b0a-18f7-4805-b2f6-40912935034a Version: $LATEST
2024-08-31T23:39:45.313Z 5c012b0a-18f7-4805-b2f6-40912935034a INFO The area is 42
2024-08-31T23:39:45.331Z 5c012b0a-18f7-4805-b2f6-40912935034a INFO CloudWatch log
 group:  /aws/lambda/myLambdaFunction
END RequestId: 5c012b0a-18f7-4805-b2f6-40912935034a
REPORT RequestId: 5c012b0a-18f7-4805-b2f6-40912935034a Duration: 20.67 ms Billed
 Duration: 21 ms Memory Size: 128 MB Max Memory Used: 66 MB Init Duration: 163.87 ms

Request ID
5c012b0a-18f7-4805-b2f6-40912935034a
```

Python

```
Status: Succeeded
Test Event Name: myTestEvent

Response
"{\"area\": 42}"

Function Logs
START RequestId: 2d0b1579-46fb-4bf7-a6e1-8e08840eae5b Version: $LATEST
The area is 42
[INFO] 2024-08-31T23:43:26.428Z 2d0b1579-46fb-4bf7-a6e1-8e08840eae5b CloudWatch logs
 group: /aws/lambda/myLambdaFunction
```

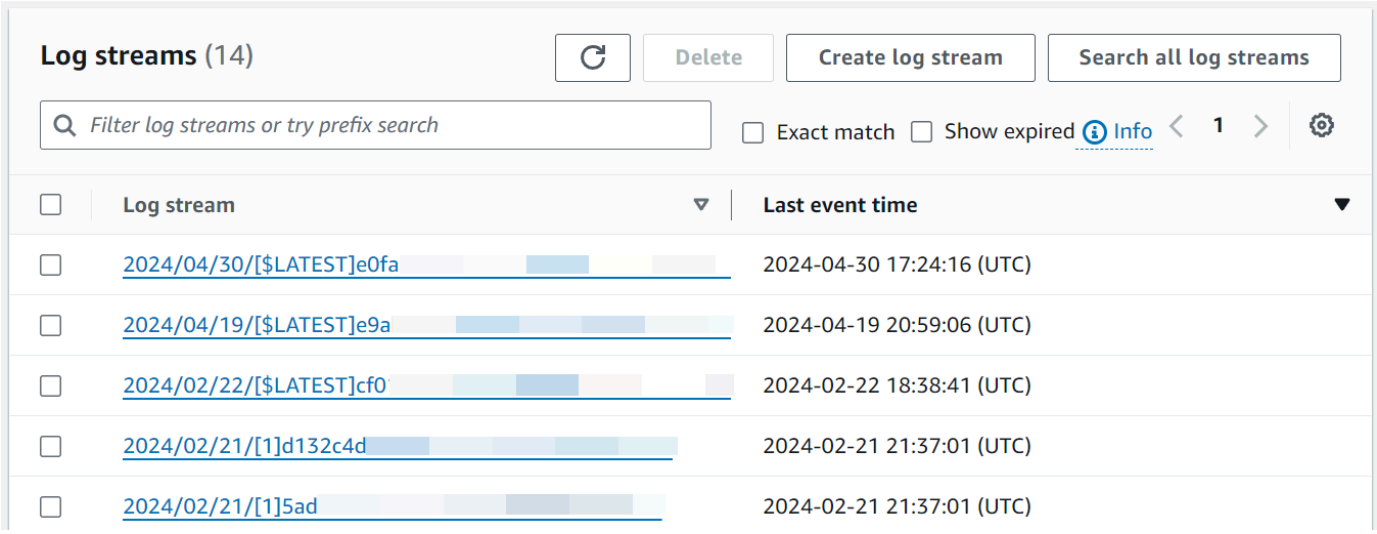
```
END RequestId: 2d0b1579-46fb-4bf7-a6e1-8e08840eae5b
REPORT RequestId: 2d0b1579-46fb-4bf7-a6e1-8e08840eae5b Duration: 1.42 ms Billed
 Duration: 2 ms Memory Size: 128 MB Max Memory Used: 39 MB Init Duration: 123.74 ms

Request ID
2d0b1579-46fb-4bf7-a6e1-8e08840eae5b
```

Saat Anda memanggil fungsi Anda di luar konsol Lambda, Anda harus menggunakan CloudWatch Log untuk melihat hasil eksekusi fungsi Anda.

Untuk melihat catatan pemanggilan fungsi Anda di CloudWatch Log

1. Buka Grup log halaman dari CloudWatch konsol.
2. Pilih grup log untuk fungsi Anda (`/aws/lambda/myLambdaFunction`). Ini adalah nama grup log yang dicetak fungsi Anda ke konsol.
3. Gulir ke bawah dan pilih Log aliran untuk pemanggilan fungsi yang ingin Anda lihat.



Anda akan melihat output yang serupa dengan yang berikut:

Node.js

```
INIT_START Runtime Version: nodejs:22.v13    Runtime Version ARN:
 arn:aws:lambda:us-
west-2::runtime:e3aaabf6b92ef8755eaae2f4bfdcb7eb8c4536a5e044900570a42bdba7b869d9
START RequestId: aba6c0fc-cf99-49d7-a77d-26d805dacd20 Version: $LATEST
2024-08-23T22:04:15.809Z    5c012b0a-18f7-4805-b2f6-40912935034a  INFO The area
 is 42
```

```
2024-08-23T22:04:15.810Z    aba6c0fc-cf99-49d7-a77d-26d805dacd20  INFO
 CloudWatch log group:  /aws/lambda/myLambdaFunction
END RequestId: aba6c0fc-cf99-49d7-a77d-26d805dacd20
REPORT RequestId: aba6c0fc-cf99-49d7-a77d-26d805dacd20    Duration: 17.77 ms
 Billed Duration: 18 ms    Memory Size: 128 MB    Max Memory Used: 67 MB    Init
 Duration: 178.85 ms
```

Python

```
INIT_START Runtime Version: python:3.13.v16    Runtime Version ARN:
 arn:aws:lambda:us-
west-2::runtime:ca202755c87b9ec2b58856efb7374b4f7b655a0ea3deb1d5acc9aee9e297b072
START RequestId: 9d4096ee-acb3-4c25-be10-8a210f0a9d8e Version: $LATEST
The area is 42
[INFO] 2024-09-01T00:05:22.464Z 9315ab6b-354a-486e-884a-2fb2972b7d84 CloudWatch
 logs group: /aws/lambda/myLambdaFunction
END RequestId: 9d4096ee-acb3-4c25-be10-8a210f0a9d8e
REPORT RequestId: 9d4096ee-acb3-4c25-be10-8a210f0a9d8e    Duration: 1.15 ms
 Billed Duration: 2 ms    Memory Size: 128 MB    Max Memory Used: 40 MB
```

Bersihkan

Ketika Anda selesai bekerja dengan fungsi contoh, hapus. Anda juga dapat menghapus grup log yang menyimpan log fungsi, dan peran eksekusi yang dibuat konsol.

Untuk menghapus Lambda Event Function

1. Buka Halaman Fungsi Acara dari konsol Lambda.
2. Pilih Fungsi Event yang Anda buat.
3. Pilih Tindakan, Hapus.
4. Mengetik **`confirm`** di bidang input teks dan pilih Hapus.

Untuk menghapus grup log

1. Buka Grup log halaman dari CloudWatch konsol.
2. Pilih grup log fungsi (`/aws/lambda/myLambdaFunction`).
3. Pilih Tindakan, Hapus grup log.
4. Di kotak dialog Hapus grup log, pilih Hapus.

Untuk menghapus peran eksekusi

1. Buka Halaman Peran dari AWS Identity and Access Management (IAM) konsol.
2. Pilih peran eksekusi fungsi (misalnya, `myLambdaFunction-role-31exxmpl`).
3. Pilih Hapus.
4. Di dalam Hapus peran kotak dialog, masukkan nama peran, lalu pilih Hapus.

Sumber daya tambahan dan langkah selanjutnya

Sekarang setelah Anda membuat dan menguji fungsi Lambda sederhana menggunakan konsol, lakukan langkah-langkah berikut ini:

- Pelajari cara menambahkan dependensi ke fungsi Anda dan menyebarkannya menggunakan paket penerapan.zip. Pilih bahasa pilihan Anda dari tautan berikut.

  Node.js

  the section called “Deploy arsip file .zip”

  Typescript

  the section called “Deploy arsip file .zip”

  Python

  the section called “Deploy arsip file .zip”

  Ruby

  the section called “Deploy arsip file .zip”

  Java

  the section called “Deploy arsip file .zip”

  Go

  the section called “Deploy arsip file .zip”

  C#

  the section called “Paket deployment”

- Untuk mempelajari cara memanggil fungsi Lambda menggunakan fungsi lain Layanan AWS, lihat Tutorial: Menggunakan pemicu Amazon S3 untuk menjalankan Fungsi Peristiwa Lambda.

- Pilih salah satu tutorial berikut untuk contoh yang lebih kompleks menggunakan Lambda dengan yang lain Layanan AWS.
  - Tutorial: Menggunakan Lambda dengan API Gateway: Buat API REST API Gateway Amazon yang memanggil fungsi Lambda.
  - Menggunakan fungsi Lambda untuk mengakses database Amazon RDS: Gunakan fungsi Lambda untuk menulis data ke database Amazon Relational Database Service (Amazon RDS) melalui RDS Proxy.
  - Menggunakan pemicu Amazon S3 untuk membuat gambar thumbnail: Gunakan fungsi Lambda untuk membuat thumbnail setiap kali file gambar diunggah ke bucket Amazon S3.

Memulai dengan contoh aplikasi dan pola

Sumber daya berikut dapat digunakan untuk membuat dan menyebarkan aplikasi tanpa server dengan cepat yang mengimplementasikan beberapa kasus penggunaan Lambda umum. Untuk setiap contoh aplikasi, kami memberikan instruksi untuk membuat dan mengonfigurasi sumber daya secara manual menggunakan Konsol Manajemen AWS, atau menggunakan AWS Serverless Application Model untuk menyebarkan sumber daya menggunakan IaC. Ikuti petunjuk konsol untuk mempelajari lebih lanjut tentang mengonfigurasi AWS sumber daya individual untuk setiap aplikasi, atau gunakan AWS SAM untuk menyebarkan sumber daya dengan cepat seperti yang Anda lakukan di lingkungan produksi.

Pemrosesan File

- Aplikasi Enkripsi PDF : Buat aplikasi tanpa server yang mengenkripsi file PDF saat diunggah ke bucket Amazon Simple Storage Service dan menyimpannya ke bucket lain, yang berguna untuk mengamankan dokumen sensitif saat diunggah.

- Aplikasi Analisis Gambar : Buat aplikasi tanpa server yang mengekstrak teks dari gambar menggunakan Amazon Rekognition, yang berguna untuk pemrosesan dokumen, moderasi konten, dan analisis gambar otomatis.

Integrasi Database

- Queue-to-Database Aplikasi : Buat aplikasi tanpa server yang menulis pesan antrian ke database Amazon RDS, yang berguna untuk memproses pendaftaran pengguna dan menangani pengiriman pesanan.

- Database Event Handler : Buat aplikasi tanpa server yang merespons perubahan tabel Amazon DynamoDB, yang berguna untuk pencatatan audit, replikasi data, dan alur kerja otomatis.

Tugas Terjadwal

- Aplikasi Pemeliharaan Database : Buat aplikasi tanpa server yang secara otomatis menghapus entri yang lebih lama dari 12 bulan dari tabel Amazon DynamoDB pada jadwal cron. Ini berguna untuk pemeliharaan database otomatis dan manajemen siklus hidup data.
- Buat EventBridge jadwal Penjadwal untuk fungsi Lambda : Gunakan Pen EventBridge jadwal untuk memanggil fungsi Lambda pada jadwal satu kali atau berulang. EventBridge Penjadwal mendukung ekspresi cron dan rate, jendela waktu yang fleksibel, kebijakan coba ulang, dan antrian surat mati. Anda dapat membuat jadwal langsung dari konsol Lambda menggunakan aliran pemicu Tambah, atau dari konsol Pen EventBridge jadwal.

Sumber daya tambahan

Gunakan sumber daya berikut untuk menjelajahi Lambda dan pengembangan aplikasi tanpa server lebih lanjut:

- Serverless Land : pustaka pola siap pakai untuk membangun aplikasi tanpa server. Ini membantu pengembang membuat aplikasi lebih cepat menggunakan AWS layanan seperti Lambda, API Gateway, dan EventBridge. Situs ini menawarkan solusi pra-bangun dan praktik terbaik, sehingga lebih mudah untuk mengembangkan sistem tanpa server.
- Aplikasi sampel Lambda : Aplikasi yang tersedia di GitHub repositori untuk panduan ini. Sampel ini menunjukkan penggunaan berbagai bahasa dan AWS layanan. Setiap contoh aplikasi menyertakan skrip untuk memudahkan penyebaran dan pembersihan serta sumber daya pendukung.
- Contoh kode untuk Lambda menggunakan AWS SDK : Contoh yang menunjukkan cara menggunakan Lambda dengan AWS perangkat pengembangan perangkat lunak (SDK). Contoh-

contoh ini mencakup dasar-dasar, tindakan, skenario, dan kontribusi AWS komunitas. Contohnya mencakup operasi penting, fungsi layanan individu, dan tugas tertentu menggunakan beberapa fungsi atau AWS layanan.

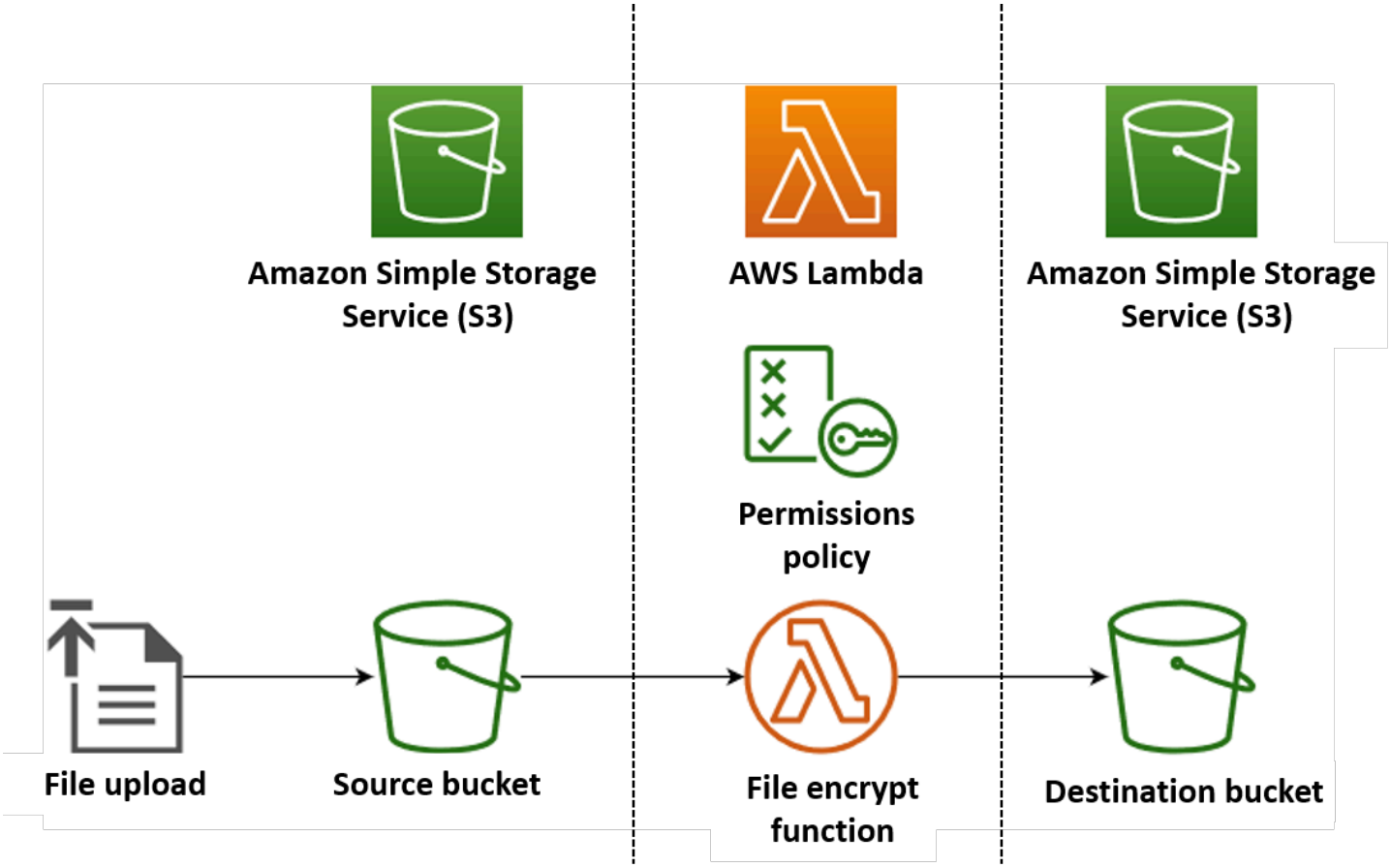
Membuat aplikasi pemrosesan file tanpa server

Salah satu kasus penggunaan yang paling umum untuk Lambda adalah melakukan tugas pemrosesan file. Misalnya, Anda mungkin menggunakan fungsi Lambda untuk secara otomatis membuat file PDF dari file HTML atau gambar, atau untuk membuat thumbnail saat pengguna mengunggah gambar.

Dalam contoh ini, Anda membuat aplikasi yang secara otomatis mengenkripsi file PDF saat diunggah ke bucket Amazon Simple Storage Service (Amazon S3). Untuk mengimplementasikan aplikasi ini, Anda membuat sumber daya berikut:

- Bucket S3 bagi pengguna untuk mengunggah file PDF ke
- Fungsi Lambda dalam Python yang membaca file yang diunggah dan membuat versi terenkripsi yang dilindungi kata sandi
- Bucket S3 kedua untuk Lambda untuk menyimpan file terenkripsi

Anda juga membuat kebijakan AWS Identity and Access Management (IAM) untuk memberikan izin fungsi Lambda Anda untuk melakukan operasi baca dan tulis pada bucket S3 Anda.



> **Tip**
>
> Jika Anda baru mengenal Lambda, kami sarankan Anda memulai dengan tutorial the section called “Buat fungsi pertama Anda” sebelum membuat aplikasi contoh ini.

Anda dapat menerapkan aplikasi secara manual dengan membuat dan mengonfigurasi resource dengan Konsol Manajemen AWS atau AWS Command Line Interface ()AWS CLI. Anda juga dapat menerapkan aplikasi dengan menggunakan AWS Serverless Application Model (AWS SAM). AWS SAM adalah alat infrastruktur sebagai kode (IAc). Dengan IAc, Anda tidak membuat sumber daya secara manual, tetapi mendefinisikannya dalam kode dan kemudian menerapkannya secara otomatis.

Jika Anda ingin mempelajari lebih lanjut tentang menggunakan Lambda dengan IAc sebelum menerapkan aplikasi contoh ini, lihat. the section called “Infrastruktur sebagai kode (IAc)”

Buat file kode sumber fungsi Lambda

Buat file berikut di direktori proyek Anda:

- `lambda_function.py`- kode fungsi Python untuk fungsi Lambda yang melakukan enkripsi file
- `requirements.txt`- file manifes yang mendefinisikan dependensi yang dibutuhkan kode fungsi Python Anda

Perluas bagian berikut untuk melihat kode dan untuk mempelajari lebih lanjut tentang peran setiap file. Untuk membuat file di komputer lokal Anda, salin dan tempel kode di bawah ini, atau unduh file dari repo GitHub aws-lambda-developer-guide.

Kode fungsi Python

Salin dan tempel kode berikut ke dalam file bernama`lambda_function.py`.

```
from pypdf import PdfReader, PdfWriter
import uuid
import os
from urllib.parse import unquote_plus
import boto3

# Create the S3 client to download and upload objects from S3
s3_client = boto3.client('s3')

def lambda_handler(event, context):
    # Iterate over the S3 event object and get the key for all uploaded files
    for record in event['Records']:
        bucket = record['s3']['bucket']['name']
        key = unquote_plus(record['s3']['object']['key']) # Decode the S3 object key to
 remove any URL-encoded characters
        download_path = f'/tmp/{uuid.uuid4()}.pdf' # Create a path in the Lambda tmp
 directory to save the file to
        upload_path = f'/tmp/converted-{uuid.uuid4()}.pdf' # Create another path to
 save the encrypted file to

        # If the file is a PDF, encrypt it and upload it to the destination S3 bucket
        if key.lower().endswith('.pdf'):
            s3_client.download_file(bucket, key, download_path)
            encrypt_pdf(download_path, upload_path)
            encrypted_key = add_encrypted_suffix(key)
            s3_client.upload_file(upload_path, f'{bucket}-encrypted', encrypted_key)
```

```
# Define the function to encrypt the PDF file with a password
def encrypt_pdf(file_path, encrypted_file_path):
    reader = PdfReader(file_path)
    writer = PdfWriter()

    for page in reader.pages:
        writer.add_page(page)

    # Add a password to the new PDF
    writer.encrypt("my-secret-password")

    # Save the new PDF to a file
    with open(encrypted_file_path, "wb") as file:
        writer.write(file)

# Define a function to add a suffix to the original filename after encryption
def add_encrypted_suffix(original_key):
    filename, extension = original_key.rsplit('.', 1)
    return f'{filename}_encrypted.{extension}'
```

> **Note**
>
> Dalam contoh kode ini, kata sandi untuk file terenkripsi (`my-secret-password`) di-hardcode ke dalam kode fungsi. Dalam aplikasi produksi, jangan sertakan informasi sensitif seperti kata sandi dalam kode fungsi Anda. Sebagai gantinya, buat AWS Secrets Manager rahasia dan kemudian gunakan ekstensi AWS Parameter dan Rahasia Lambda untuk mengambil kredensil Anda di Fungsi Acara Lambda Anda.

Kode fungsi python berisi tiga fungsi - fungsi handler yang dijalankan Lambda saat fungsi Anda dipanggil, dan dua fungsi terpisah bernama `add_encrypted_suffix` dan `encrypt_pdf` yang dipanggil handler untuk melakukan enkripsi PDF.

Saat fungsi Anda dipanggil oleh Amazon S3, Lambda meneruskan argumen peristiwa berformat JSON ke fungsi yang berisi detail tentang peristiwa yang menyebabkan pemanggilan. Dalam hal ini, informasi termasuk nama bucket S3 dan kunci objek untuk file yang diunggah. Untuk mempelajari lebih lanjut tentang format objek acara untuk Amazon S3, lihat. the section called “S3”

Fungsi Anda kemudian menggunakan file AWS SDK untuk Python (Boto3) untuk mengunduh file PDF yang ditentukan dalam objek acara ke direktori penyimpanan sementara lokalnya, sebelum mengenkripsi mereka menggunakan pustaka pypdf.

Terakhir, fungsi ini menggunakan Boto3 SDK untuk menyimpan file terenkripsi di bucket tujuan S3 Anda.

`file manifes requirements.txt`

Salin dan tempel kode berikut ke dalam file bernama`requirements.txt`.

```
boto3
pypdf
```

Untuk contoh ini, kode fungsi Anda hanya memiliki dua dependensi yang bukan bagian dari pustaka Python standar - SDK for Python (Boto3) dan paket yang digunakan fungsi untuk melakukan enkripsi PDF. `pypdf`

> **Note**
>
> Versi SDK untuk Python (Boto3) disertakan sebagai bagian dari runtime Lambda, sehingga kode Anda akan berjalan tanpa menambahkan Boto3 ke paket penerapan fungsi Anda. Namun, untuk mempertahankan kontrol penuh dependensi fungsi Anda dan menghindari kemungkinan masalah dengan misalignment versi, praktik terbaik untuk Python adalah menyertakan semua dependensi fungsi dalam paket penerapan fungsi Anda. Lihat the section called “Dependensi runtime dengan Python” untuk mempelajari selengkapnya.

Terapkan aplikasi

Anda dapat membuat dan menerapkan sumber daya untuk aplikasi contoh ini baik secara manual atau dengan menggunakan AWS SAM. Dalam lingkungan produksi, kami menyarankan Anda menggunakan alat IAC AWS SAM untuk menyebarkan seluruh aplikasi tanpa server dengan cepat dan berulang tanpa menggunakan proses manual.

Menyebarkan sumber daya secara manual

Untuk menerapkan aplikasi Anda secara manual:

- Buat bucket Amazon S3 sumber dan tujuan
- Buat fungsi Lambda yang mengenkripsi file PDF dan menyimpan versi terenkripsi ke bucket S3
- Konfigurasikan pemicu Lambda yang memanggil fungsi Anda saat objek diunggah ke bucket sumber Anda

Sebelum memulai, pastikan Python diinstal pada mesin build Anda.

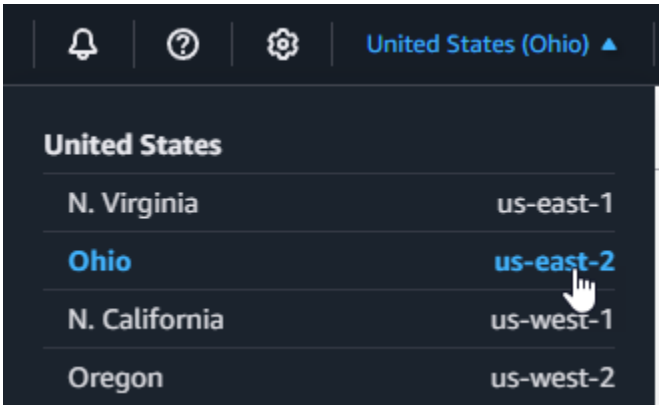
Buat dua ember S3

Pertama buat dua ember S3. Bucket pertama adalah bucket sumber tempat Anda mengunggah file PDF Anda. Bucket kedua digunakan oleh Lambda untuk menyimpan file terenkripsi saat Anda menjalankan fungsi.

Console

Untuk membuat bucket S3 (konsol)

1. Buka halaman bucket tujuan umum konsol Amazon S3.
2. Pilih yang Wilayah AWS paling dekat dengan lokasi geografis Anda. Anda dapat mengubah wilayah Anda menggunakan daftar drop-down di bagian atas layar.



3. Pilih Buat bucket.
4. Pada Konfigurasi umum, lakukan hal berikut:
   a. Untuk jenis Bucket, pastikan Tujuan umum dipilih.
   b. Untuk nama Bucket, masukkan nama unik global yang memenuhi aturan penamaan bucket Amazon S3. Nama bucket hanya dapat berisi huruf kecil, angka, titik (.), dan tanda hubung (-).
5. Biarkan semua opsi lain disetel ke nilai defaultnya dan pilih Buat bucket.

6. Ulangi langkah 1 hingga 4 untuk membuat bucket tujuan Anda. Untuk nama Bucket**amzn-s3-demo-bucket-encrypted**, masukkan, di **amzn-s3-demo-bucket** mana nama bucket sumber yang baru saja Anda buat.

AWS CLI

Sebelum Anda mulai, pastikan bahwa AWS CLI diinstal pada mesin build Anda.

Untuk membuat ember Amazon S3 (AWS CLI)

1. Jalankan perintah CLI berikut untuk membuat bucket sumber Anda. Nama yang Anda pilih untuk bucket Anda harus unik secara global dan ikuti aturan penamaan bucket Amazon S3. Nama hanya dapat berisi huruf kecil, angka, titik (.), dan tanda hubung (-). Untuk `region` dan`LocationConstraint`, pilih yang paling Wilayah AWSdekat dengan lokasi geografis Anda.

   ```
   aws s3api create-bucket --bucket amzn-s3-demo-bucket --region us-east-2 \
   --create-bucket-configuration LocationConstraint=us-east-2
   ```

   Kemudian dalam tutorial, Anda harus membuat fungsi Lambda Anda Wilayah AWS sama dengan bucket sumber Anda, jadi catat wilayah yang Anda pilih.
2. Jalankan perintah berikut untuk membuat bucket tujuan Anda. Untuk nama bucket, Anda harus menggunakan**amzn-s3-demo-bucket-encrypted**, di **amzn-s3-demo-bucket** mana nama bucket sumber yang Anda buat di langkah 1. Untuk `region` dan`LocationConstraint`, pilih yang sama dengan yang Wilayah AWS Anda gunakan untuk membuat bucket sumber Anda.

   ```
   aws s3api create-bucket --bucket amzn-s3-demo-bucket-encrypted --region us-
   east-2 \
   --create-bucket-configuration LocationConstraint=us-east-2
   ```

Membuat peran eksekusi

Peran eksekusi adalah peran IAM yang memberikan izin fungsi Lambda untuk mengakses dan sumber daya. Layanan AWS Untuk memberikan akses baca dan tulis fungsi Anda ke Amazon S3, Anda melampirkan kebijakan AWS terkelola. `AmazonS3FullAccess`

Console

Untuk membuat peran eksekusi dan melampirkan kebijakan **AmazonS3FullAccess**terkelola (konsol)

1. Buka Halaman peran di konsol IAM.
2. Pilih Buat peran.
3. Untuk jenis entitas Tepercaya, pilih AWS layanan, dan untuk kasus Penggunaan, pilih Lambda.
4. Pilih Berikutnya.
5. Tambahkan kebijakan `AmazonS3FullAccess` terkelola dengan melakukan hal berikut:
   a. Dalam kebijakan Izin, masukkan **AmazonS3FullAccess** ke dalam bilah pencarian.
   b. Pilih kotak centang di sebelah kebijakan.
   c. Pilih Berikutnya.
6. Dalam Rincian Peran, untuk nama Peran masukkan**LambdaS3Role**.
7. Pilih Buat Peran.

AWS CLI

Untuk membuat peran eksekusi dan melampirkan kebijakan **AmazonS3FullAccess**terkelola (AWS CLI)

1. Simpan JSON berikut dalam file bernama`trust-policy.json`. Kebijakan kepercayaan ini memungkinkan Lambda untuk menggunakan izin peran dengan memberikan `lambda.amazonaws.com` izin utama layanan untuk memanggil tindakan AWS Security Token Service ()AWS STS. `AssumeRole`

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "lambda.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
```

```
    }
  ]
}
```

2. Dari direktori tempat Anda menyimpan dokumen kebijakan kepercayaan JSON, jalankan perintah CLI berikut untuk membuat peran eksekusi.

```
aws iam create-role --role-name LambdaS3Role --assume-role-policy-document
 file://trust-policy.json
```

3. Untuk melampirkan kebijakan `AmazonS3FullAccess` terkelola, jalankan perintah CLI berikut.

```
aws iam attach-role-policy --role-name LambdaS3Role --policy-arn
 arn:aws:iam::aws:policy/AmazonS3FullAccess
```

Buat paket penerapan fungsi

Untuk membuat fungsi Anda, Anda membuat paket deployment yang berisi kode fungsi dan dependensinya. Untuk aplikasi ini, kode fungsi Anda menggunakan pustaka terpisah untuk enkripsi PDF.

Untuk membuat paket deployment

1. Arahkan ke direktori proyek yang berisi `lambda_function.py` dan `requirements.txt` file yang Anda buat atau unduh dari GitHub sebelumnya dan buat direktori baru bernama`package`.
2. Instal dependensi yang ditentukan dalam `requirements.txt` file di `package` direktori Anda dengan menjalankan perintah berikut.

```
pip install -r requirements.txt --target ./package/
```

3. Buat file.zip yang berisi kode aplikasi Anda dan dependensinya. Di Linux atau macOS, jalankan perintah berikut dari antarmuka baris perintah Anda.

```
cd package
zip -r ../lambda_function.zip .
cd ..
zip lambda_function.zip lambda_function.py
```

Di Windows, gunakan alat zip pilihan Anda untuk membuat file `lambda_function.zip`. Pastikan bahwa `lambda_function.py` file Anda dan folder yang berisi dependensi Anda semuanya berada di root file.zip.

Anda juga dapat membuat paket deployment menggunakan lingkungan virtual Python. Lihat Bekerja dengan arsip file.zip untuk Fungsi Acara Python Lambda

Buat fungsi Lambda

Anda sekarang menggunakan paket penyebaran yang Anda buat pada langkah sebelumnya untuk menyebarkan fungsi Lambda Anda.

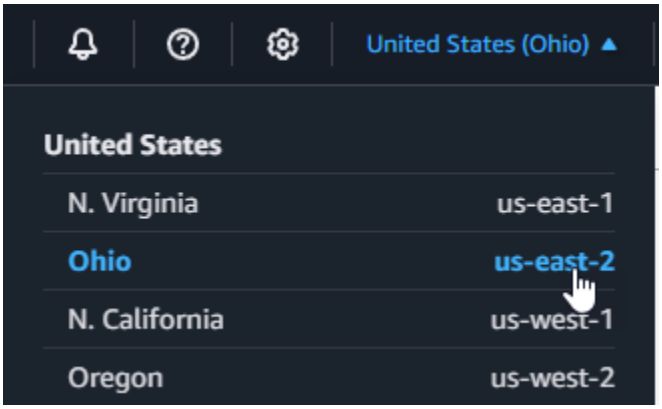
Console

Untuk membuat fungsi (konsol)

Untuk membuat fungsi Lambda menggunakan konsol, pertama-tama Anda membuat fungsi dasar yang berisi beberapa kode 'Hello world'. Anda kemudian mengganti kode ini dengan kode fungsi Anda sendiri dengan mengunggah file the.zip yang Anda buat pada langkah sebelumnya.

Untuk memastikan bahwa fungsi Anda tidak habis saat mengenkripsi file PDF besar, Anda mengonfigurasi pengaturan memori dan batas waktu fungsi. Anda juga mengatur format log fungsi ke JSON. Mengkonfigurasi log berformat JSON diperlukan saat menggunakan skrip pengujian yang disediakan sehingga dapat membaca status pemanggilan fungsi dari CloudWatch Log untuk mengonfirmasi pemanggilan yang berhasil.

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pastikan Anda bekerja di tempat yang sama saat Wilayah AWS Anda membuat bucket S3. Anda dapat mengubah wilayah Anda menggunakan daftar drop-down di bagian atas layar.



3. Pilih Buat Fungsi Acara.
4. Pilih Tulis dari awal.
5. Di bagian Informasi dasar, lakukan hal berikut:
    a. Untuk Nama fungsi, masukkan **`EncryptPDF`**.
    b. Untuk Runtime pilih Python 3.12.
    c. Untuk Arsitektur, pilih x86_64.
6. Lampirkan peran eksekusi yang Anda buat pada langkah sebelumnya dengan melakukan hal berikut:
    a. Perluas bagian Ubah peran eksekusi default.
    b. Pilih Gunakan peran yang ada.
    c. Di bawah Peran yang ada, pilih peran Anda (`LambdaS3Role`).
7. Pilih Buat Fungsi Acara.

Untuk mengunggah kode fungsi (konsol)

1. Di panel Sumber kode, pilih Unggah dari.
2. Pilih file.zip.
3. Pilih Unggah.
4. Di pemilih file, pilih file.zip Anda dan pilih Buka.
5. Pilih Simpan.

Untuk mengkonfigurasi memori fungsi dan batas waktu (konsol)

1. Pilih tab Konfigurasi untuk fungsi Anda.
2. Di panel konfigurasi Umum, pilih Edit.
3. Atur Memori ke 256 MB dan Timeout menjadi 15 detik.
4. Pilih Simpan.

Untuk mengkonfigurasi format log (konsol)

1. Pilih tab Konfigurasi untuk fungsi Anda.
2. Pilih Alat pemantauan dan operasi.

3. Di panel konfigurasi Logging, pilih Edit.
4. Untuk konfigurasi Logging, pilih JSON.
5. Pilih Simpan.

AWS CLI

Untuk membuat fungsi (AWS CLI)

- Jalankan perintah berikut dari direktori yang berisi `lambda_function.zip` file Anda.Untuk `region` parameternya, ganti `us-east-2` dengan wilayah tempat Anda membuat bucket S3.

```
aws lambda create-function --function-name EncryptPDF \
--zip-file fileb://lambda_function.zip --handler lambda_function.lambda_handler
 \
--runtime python3.12 --timeout 15 --memory-size 256 \
--role arn:aws:iam::123456789012:role/LambdaS3Role --region us-east-2 \
--logging-config LogFormat=JSON
```

Konfigurasikan pemicu Amazon S3 untuk menjalankan fungsi

Agar fungsi Lambda dapat berjalan saat mengunggah file ke bucket sumber, Anda perlu mengonfigurasi pemicu untuk fungsi Anda. Anda dapat mengonfigurasi pemicu Amazon S3 menggunakan konsol atau. AWS CLI

> **Important**
>
> Prosedur ini mengonfigurasi bucket S3 untuk memanggil fungsi Anda setiap kali objek dibuat di bucket. Pastikan untuk mengonfigurasi ini hanya di bucket sumber. Jika fungsi Lambda Anda membuat objek dalam bucket yang sama yang memanggilnya, fungsi Anda dapat dipanggil terus menerus dalam satu loop. Hal ini dapat mengakibatkan biaya yang tidak diharapkan ditagih ke Anda Akun AWS.

Console

Untuk mengonfigurasi pemicu Amazon S3 (konsol)

1. Buka halaman Fungsi Acara konsol Lambda dan pilih fungsi Anda ()`EncryptPDF`.

2. Pilih Tambahkan pemicu.
3. Pilih S3.
4. Di bawah Bucket, pilih bucket sumber Anda.
5. Di bawah Jenis acara, pilih Semua objek membuat acara.
6. Di bawah Pemanggilan rekursif, pilih kotak centang untuk mengetahui bahwa tidak disarankan menggunakan bucket S3 yang sama untuk input dan output. Anda dapat mempelajari lebih lanjut tentang pola pemanggilan rekursif di Lambda dengan membaca pola rekursif yang menyebabkan fungsi Lambda yang tidak terkendali di Tanah Tanpa Server.
7. Pilih Tambahkan.

   Saat Anda membuat pemicu menggunakan konsol Lambda, Lambda secara otomatis membuat kebijakan berbasis sumber daya untuk memberikan layanan yang Anda pilih izin untuk menjalankan fungsi Anda.

AWS CLI

Untuk mengonfigurasi pemicu Amazon S3 (AWS CLI)

1. Tambahkan kebijakan berbasis sumber daya ke fungsi yang memungkinkan bucket sumber Amazon S3 menjalankan fungsi saat menambahkan file. Pernyataan kebijakan berbasis sumber daya memberikan Layanan AWS izin lain untuk menjalankan fungsi Anda. Untuk memberikan izin Amazon S3 untuk menjalankan fungsi Anda, jalankan perintah CLI berikut. Pastikan untuk mengganti `source-account` parameter dengan Akun AWS ID Anda sendiri dan menggunakan nama bucket sumber Anda sendiri.

   ```
   aws lambda add-permission --function-name EncryptPDF \
   --principal s3.amazonaws.com --statement-id s3invoke --action
    "lambda:InvokeFunction" \
   --source-arn arn:aws:s3:::amzn-s3-demo-bucket \
   --source-account 123456789012
   ```

   Kebijakan yang Anda tetapkan dengan perintah ini memungkinkan Amazon S3 untuk menjalankan fungsi Anda hanya ketika tindakan dilakukan di bucket sumber Anda.

> **Note**
>
> Meskipun nama bucket S3 unik secara global, saat menggunakan kebijakan berbasis sumber daya, praktik terbaik adalah menentukan bahwa bucket harus menjadi milik akun Anda. Ini karena jika Anda menghapus bucket, Anda dapat membuat bucket dengan Amazon Resource Name (ARN) yang sama. Akun AWS

2. Simpan JSON berikut dalam file bernama`notification.json`. Saat diterapkan ke bucket sumber Anda, JSON ini mengonfigurasi bucket untuk mengirim notifikasi ke fungsi Lambda Anda setiap kali objek baru ditambahkan. Ganti Akun AWS nomor dan Wilayah AWS dalam fungsi Lambda ARN dengan nomor akun dan wilayah Anda sendiri.

```
{
"LambdaFunctionConfigurations": [
    {
      "Id": "EncryptPDFEventConfiguration",
      "LambdaFunctionArn": "arn:aws:lambda:us-
east-2:123456789012:function:EncryptPDF",
      "Events": [ "s3:ObjectCreated:Put" ]
    }
  ]
}
```

3. Jalankan perintah CLI berikut untuk menerapkan pengaturan notifikasi dalam file JSON yang Anda buat ke bucket sumber Anda. Ganti `amzn-s3-demo-bucket` dengan nama bucket sumber Anda sendiri.

```
aws s3api put-bucket-notification-configuration --bucket amzn-s3-demo-bucket \
--notification-configuration file://notification.json
```

Untuk mempelajari lebih lanjut tentang *`put-bucket-notification-configuration`* perintah dan *`notification-configuration`* opsi, lihat put-bucket-notification-configuration di CLI Command Reference.AWS

Menyebarkan sumber daya menggunakan AWS SAM

Sebelum memulai, pastikan bahwa Docker dan versi terbaru diinstal pada mesin build Anda. AWS SAMCLI

1. Di direktori proyek Anda, salin dan tempel kode berikut ke dalam file bernama`template.yaml`. Ganti nama bucket placeholder:

   - Untuk bucket sumber, ganti `amzn-s3-demo-bucket` dengan nama apa pun yang sesuai dengan aturan penamaan bucket S3.
   - Untuk bucket tujuan, ganti `amzn-s3-demo-bucket-encrypted` dengan`<source-bucket-name>-encrypted`, di `<source-bucket>` mana nama yang Anda pilih untuk bucket sumber Anda.

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31

Resources:
  EncryptPDFFunction:
    Type: AWS::Serverless::Function
    Properties:
      FunctionName: EncryptPDF
      Architectures: [x86_64]
      CodeUri: ./
      Handler: lambda_function.lambda_handler
      Runtime: python3.12
      Timeout: 15
      MemorySize: 256
      LoggingConfig:
        LogFormat: JSON
      Policies:
        - AmazonS3FullAccess
      Events:
        S3Event:
          Type: S3
          Properties:
            Bucket: !Ref PDFSourceBucket
            Events: s3:ObjectCreated:*

  PDFSourceBucket:
    Type: AWS::S3::Bucket
    Properties:
      BucketName: amzn-s3-demo-bucket

  EncryptedPDFBucket:
    Type: AWS::S3::Bucket
```

```
    Properties:
      BucketName: amzn-s3-demo-bucket-encrypted
```

AWS SAM Template menentukan sumber daya yang Anda buat untuk aplikasi Anda. Dalam contoh ini, template mendefinisikan fungsi Lambda menggunakan `AWS::Serverless::Function` tipe dan dua bucket S3 menggunakan tipe. `AWS::S3::Bucket` Nama bucket yang ditentukan dalam template adalah placeholder. Sebelum menerapkan aplikasi AWS SAM, Anda perlu mengedit template untuk mengganti nama bucket dengan nama unik global yang memenuhi aturan penamaan bucket S3. Langkah ini dijelaskan lebih lanjut dalamthe section called “Menyebarkan sumber daya menggunakan AWS SAM”.

Definisi sumber daya fungsi Lambda mengonfigurasi pemicu untuk fungsi menggunakan properti event. `S3Event` Pemicu ini menyebabkan fungsi Anda dipanggil setiap kali objek dibuat di bucket sumber Anda.

Definisi fungsi juga menentukan kebijakan AWS Identity and Access Management (IAM) yang akan dilampirkan ke peran eksekusi fungsi. Kebijakan AWS terkelola `AmazonS3FullAccess` memberi fungsi Anda izin yang diperlukan untuk membaca dan menulis objek ke Amazon S3.

2. Jalankan perintah berikut dari direktori tempat Anda menyimpan`template.yaml`,`lambda_function.py`, dan `requirements.txt` file Anda.

```
sam build --use-container
```

Perintah ini mengumpulkan artefak build untuk aplikasi Anda dan menempatkannya dalam format dan lokasi yang tepat untuk menerapkannya. Menentukan `--use-container` opsi membangun fungsi Anda di dalam wadah Lambda-like Docker. Kami menggunakannya di sini sehingga Anda tidak perlu menginstal Python 3.12 di mesin lokal Anda agar build berfungsi.

Selama proses build, AWS SAM cari kode fungsi Lambda di lokasi yang Anda tentukan dengan `CodeUri` properti di template. Dalam hal ini, kami menentukan direktori saat ini sebagai location (`./`).

Jika ada `requirements.txt` file, AWS SAM gunakan untuk mengumpulkan dependensi yang ditentukan. Secara default, AWS SAM buat paket deployment .zip dengan kode fungsi dan dependensi Anda. Anda juga dapat memilih untuk menerapkan fungsi Anda sebagai gambar kontainer menggunakan PackageTypeproperti.

3. Untuk menerapkan aplikasi Anda dan membuat sumber daya Lambda dan Amazon S3 yang ditentukan dalam template AWS SAM Anda, jalankan perintah berikut.

```
sam deploy --guided
```

Menggunakan `--guided` tanda berarti itu AWS SAM akan menunjukkan kepada Anda petunjuk untuk memandu Anda melalui proses penerapan. Untuk penerapan ini, terima opsi default dengan menekan Enter.

Selama proses penerapan, AWS SAM buat sumber daya berikut di: Akun AWS

- Sebuah CloudFormation tumpukan bernama `sam-app`
- Fungsi Lambda dengan nama `EncryptPDF`
- Dua ember S3 dengan nama yang Anda pilih saat Anda mengedit file template `template.yaml` AWS SAM
- Peran eksekusi IAM untuk fungsi Anda dengan format nama `sam-app-EncryptPDFFunctionRole-2qGaapHFWOQ8`

Ketika AWS SAM selesai membuat sumber daya Anda, Anda akan melihat pesan berikut:

```
Successfully created/updated stack - sam-app in us-east-2
```

Tes aplikasi

Untuk menguji aplikasi Anda, unggah file PDF ke bucket sumber Anda, dan konfirmasikan bahwa Lambda membuat versi file terenkripsi di bucket tujuan Anda. Dalam contoh ini, Anda dapat menguji ini secara manual menggunakan konsol atau AWS CLI, atau dengan menggunakan skrip pengujian yang disediakan.

Untuk aplikasi produksi, Anda dapat menggunakan metode dan teknik pengujian tradisional, seperti pengujian unit, untuk mengonfirmasi berfungsinya kode fungsi Lambda Anda dengan benar. Praktik terbaik juga melakukan pengujian seperti yang ada di skrip pengujian yang disediakan yang melakukan pengujian integrasi dengan sumber daya berbasis cloud yang nyata. Pengujian integrasi di cloud mengonfirmasi bahwa infrastruktur Anda telah digunakan dengan benar dan bahwa peristiwa mengalir di antara berbagai layanan seperti yang diharapkan. Untuk mempelajari selengkapnya, lihat the section called “Pengujian”.

Menguji aplikasi secara manual

Anda dapat menguji fungsi Anda secara manual dengan menambahkan file PDF ke bucket sumber Amazon S3 Anda. Saat Anda menambahkan file ke bucket sumber, fungsi Lambda Anda akan dipanggil secara otomatis dan harus menyimpan versi file terenkripsi di bucket target Anda.

Console

Untuk menguji aplikasi Anda dengan mengunggah file (konsol)

1. Untuk mengunggah file PDF ke bucket S3 Anda, lakukan hal berikut:
   a. Buka halaman Bucket di konsol Amazon S3 dan pilih bucket sumber Anda.
   b. Pilih Unggah.
   c. Pilih Tambahkan file dan gunakan pemilih file untuk memilih file PDF yang ingin Anda unggah.
   d. Pilih Buka, lalu pilih Unggah.
2. Verifikasi bahwa Lambda telah menyimpan versi terenkripsi file PDF Anda di bucket target Anda dengan melakukan hal berikut:
   a. Arahkan kembali ke halaman Bucket di konsol Amazon S3 dan pilih bucket tujuan Anda.
   b. Di panel Objects, Anda sekarang akan melihat file dengan format nama `filename_encrypted.pdf` (di mana `filename.pdf` nama file yang Anda unggah ke bucket sumber Anda). Untuk mengunduh PDF terenkripsi Anda, pilih file, lalu pilih Unduh.
   c. Konfirmasikan bahwa Anda dapat membuka file yang diunduh dengan kata sandi fungsi Lambda Anda melindunginya dengan ()`my-secret-password`.

AWS CLI

Untuk menguji aplikasi Anda dengan mengunggah file (AWS CLI)

1. Dari direktori yang berisi file PDF yang ingin Anda unggah, jalankan perintah CLI berikut. Ganti `--bucket` parameter dengan nama bucket sumber Anda. Untuk `--body` parameter `--key` dan, gunakan nama file file pengujian Anda.

```
aws s3api put-object --bucket amzn-s3-demo-bucket --key test.pdf --body ./
test.pdf
```

2. Verifikasi bahwa fungsi Anda telah membuat versi terenkripsi dari file Anda dan menyimpannya ke bucket S3 target Anda. Jalankan perintah CLI berikut, ganti `amzn-s3-demo-bucket-encrypted` dengan nama bucket tujuan Anda sendiri.

```
aws s3api list-objects-v2 --bucket amzn-s3-demo-bucket-encrypted
```

Jika fungsi Anda berjalan dengan sukses, Anda akan melihat output yang mirip dengan berikut ini. Bucket target Anda harus berisi file dengan format nama*<your_test_file>*`_encrypted.pdf`, di `<your_test_file>` mana nama file yang Anda unggah.

```
{
    "Contents": [
        {
            "Key": "test_encrypted.pdf",
            "LastModified": "2023-06-07T00:15:50+00:00",
            "ETag": "\"7781a43e765a8301713f533d70968a1e\"",
            "Size": 2763,
            "StorageClass": "STANDARD"
        }
    ]
}
```

3. Untuk mengunduh file yang disimpan Lambda di bucket tujuan Anda, jalankan perintah CLI berikut. Ganti `--bucket` parameter dengan nama bucket tujuan Anda. Untuk `--key` parameternya, gunakan nama file*<your_test_file>*`_encrypted.pdf`, di `<your_test_file>` mana nama file pengujian yang Anda unggah.

```
aws s3api get-object --bucket amzn-s3-demo-bucket-encrypted --
key test_encrypted.pdf my_encrypted_file.pdf
```

Perintah ini mengunduh file ke direktori Anda saat ini dan menyimpannya sebagai`my_encrypted_file.pdf`.

4. Konfirmasikan bahwa Anda dapat membuka file yang diunduh dengan kata sandi fungsi Lambda Anda melindunginya dengan ()`my-secret-password`.

Menguji aplikasi dengan skrip otomatis

Buat file berikut di direktori proyek Anda:

- `test_pdf_encrypt.py`- skrip pengujian yang dapat Anda gunakan untuk menguji aplikasi Anda secara otomatis
- `pytest.ini`- file konfigurasi untuk skrip pengujian

Perluas bagian berikut untuk melihat kode dan untuk mempelajari lebih lanjut tentang peran setiap file.

Skrip pengujian otomatis

Salin dan tempel kode berikut ke dalam file bernama`test_pdf_encrypt.py`. Pastikan untuk mengganti nama bucket placeholder:

- Dalam `test_source_bucket_available` fungsinya, ganti `amzn-s3-demo-bucket` dengan nama bucket sumber Anda.
- Dalam `test_encrypted_file_in_bucket` fungsinya, ganti `amzn-s3-demo-bucket-encrypted` dengan*`source-bucket`*`-encrypted`, di `source-bucket>` mana nama bucket sumber Anda.
- Dalam `cleanup` fungsi, ganti `amzn-s3-demo-bucket` dengan nama bucket sumber Anda, dan ganti `amzn-s3-demo-bucket-encrypted` dengan nama bucket tujuan Anda.

```
import boto3
import json
import pytest
import time
import os

@pytest.fixture
def lambda_client():
    return boto3.client('lambda')

@pytest.fixture
def s3_client():
    return boto3.client('s3')

@pytest.fixture
def logs_client():
    return boto3.client('logs')

@pytest.fixture(scope='session')
```

```
def cleanup():
    # Create a new S3 client for cleanup
    s3_client = boto3.client('s3')

    yield
    # Cleanup code will be executed after all tests have finished

    # Delete test.pdf from the source bucket
    source_bucket = 'amzn-s3-demo-bucket'
    source_file_key = 'test.pdf'
    s3_client.delete_object(Bucket=source_bucket, Key=source_file_key)
    print(f"\nDeleted {source_file_key} from {source_bucket}")

    # Delete test_encrypted.pdf from the destination bucket
    destination_bucket = 'amzn-s3-demo-bucket-encrypted'
    destination_file_key = 'test_encrypted.pdf'
    s3_client.delete_object(Bucket=destination_bucket, Key=destination_file_key)
    print(f"Deleted {destination_file_key} from {destination_bucket}")


@pytest.mark.order(1)
def test_source_bucket_available(s3_client):
    s3_bucket_name = 'amzn-s3-demo-bucket'
    file_name = 'test.pdf'
    file_path = os.path.join(os.path.dirname(__file__), file_name)

    file_uploaded = False
    try:
        s3_client.upload_file(file_path, s3_bucket_name, file_name)
        file_uploaded = True
    except:
        print("Error: couldn't upload file")

    assert file_uploaded, "Could not upload file to S3 bucket"



@pytest.mark.order(2)
def test_lambda_invoked(logs_client):

    # Wait for a few seconds to make sure the logs are available
    time.sleep(5)

    # Get the latest log stream for the specified log group
```

```
    log_streams = logs_client.describe_log_streams(
        logGroupName='/aws/lambda/EncryptPDF',
        orderBy='LastEventTime',
        descending=True,
        limit=1
    )

    latest_log_stream_name = log_streams['logStreams'][0]['logStreamName']

    # Retrieve the log events from the latest log stream
    log_events = logs_client.get_log_events(
        logGroupName='/aws/lambda/EncryptPDF',
        logStreamName=latest_log_stream_name
    )

    success_found = False
    for event in log_events['events']:
        message = json.loads(event['message'])
        status = message.get('record', {}).get('status')
        if status == 'success':
            success_found = True
            break

    assert success_found, "Lambda function execution did not report 'success' status in
 logs."

@pytest.mark.order(3)
def test_encrypted_file_in_bucket(s3_client):
    # Specify the destination S3 bucket and the expected converted file key
    destination_bucket = 'amzn-s3-demo-bucket-encrypted'
    converted_file_key = 'test_encrypted.pdf'

    try:
        # Attempt to retrieve the metadata of the converted file from the destination
 S3 bucket
        s3_client.head_object(Bucket=destination_bucket, Key=converted_file_key)
    except s3_client.exceptions.ClientError as e:
        # If the file is not found, the test will fail
        pytest.fail(f"Converted file '{converted_file_key}' not found in the
 destination bucket: {str(e)}")

def test_cleanup(cleanup):
    # This test uses the cleanup fixture and will be executed last
```

```
    pass
```

Skrip pengujian otomatis menjalankan tiga fungsi pengujian untuk mengonfirmasi pengoperasian aplikasi Anda yang benar:

- Pengujian `test_source_bucket_available` mengonfirmasi bahwa bucket sumber Anda telah berhasil dibuat dengan mengunggah file PDF pengujian ke bucket.
- Pengujian `test_lambda_invoked` menginterogasi aliran CloudWatch log Log terbaru untuk fungsi Anda guna mengonfirmasi bahwa saat Anda mengunggah file pengujian, fungsi Lambda Anda berjalan dan melaporkan keberhasilan.
- Pengujian `test_encrypted_file_in_bucket` mengonfirmasi bahwa bucket tujuan Anda berisi file terenkripsi. `test_encrypted.pdf`

Setelah semua pengujian ini berjalan, skrip menjalankan langkah pembersihan tambahan untuk menghapus `test_encrypted.pdf` file `test.pdf` dan dari bucket sumber dan tujuan Anda.

Seperti halnya AWS SAM template, nama bucket yang ditentukan dalam file ini adalah placeholder. Sebelum menjalankan pengujian, Anda perlu mengedit file ini dengan nama bucket asli aplikasi Anda. Langkah ini dijelaskan lebih lanjut dalam the section called “Menguji aplikasi dengan skrip otomatis”

File konfigurasi skrip uji

Salin dan tempel kode berikut ke dalam file bernama`pytest.ini`.

```
[pytest]
markers =
    order: specify test execution order
```

Ini diperlukan untuk menentukan urutan pengujian dalam `test_pdf_encrypt.py` skrip berjalan.

Untuk menjalankan tes lakukan hal berikut:

1. Pastikan `pytest` modul dipasang di lingkungan lokal Anda. Anda dapat menginstal `pytest` dengan menjalankan perintah berikut:

   ```
   pip install pytest
   ```

2. Simpan file PDF bernama `test.pdf` di direktori yang berisi `pytest.ini` file `test_pdf_encrypt.py` dan.
3. Buka terminal atau program shell dan jalankan perintah berikut dari direktori yang berisi file uji.

```
pytest -s -v
```

Ketika tes selesai, Anda akan melihat output seperti berikut:

```
============================================================= test session starts
 ==============================================================
platform linux -- Python 3.12.2, pytest-7.2.2, pluggy-1.0.0 -- /usr/bin/python3
cachedir: .pytest_cache
hypothesis profile 'default' -> database=DirectoryBasedExampleDatabase('/home/
pdf_encrypt_app/.hypothesis/examples')
Test order randomisation NOT enabled. Enable with --random-order or --random-order-
bucket=<bucket_type>
rootdir: /home/pdf_encrypt_app, configfile: pytest.ini
plugins: anyio-3.7.1, hypothesis-6.70.0, localserver-0.7.1, random-order-1.1.0
collected 4 items

test_pdf_encrypt.py::test_source_bucket_available PASSED
test_pdf_encrypt.py::test_lambda_invoked PASSED
test_pdf_encrypt.py::test_encrypted_file_in_bucket PASSED
test_pdf_encrypt.py::test_cleanup PASSED
Deleted test.pdf from amzn-s3-demo-bucket
Deleted test_encrypted.pdf from amzn-s3-demo-bucket-encrypted


============================================================== 4 passed in 7.32s
 ==============================================================
```

Langkah selanjutnya

Sekarang Anda telah membuat aplikasi contoh ini, Anda dapat menggunakan kode yang disediakan sebagai dasar untuk membuat jenis aplikasi pemrosesan file lainnya. Ubah kode dalam `lambda_function.py` file untuk mengimplementasikan logika pemrosesan file untuk kasus penggunaan Anda.

Banyak kasus penggunaan pemrosesan file yang khas melibatkan pemrosesan gambar. Saat menggunakan Python, perpustakaan pemrosesan gambar yang paling populer seperti bantal biasanya berisi komponen C atau C ++. Untuk memastikan bahwa paket penerapan fungsi Anda kompatibel dengan lingkungan eksekusi Lambda, penting untuk menggunakan biner distribusi sumber yang benar.

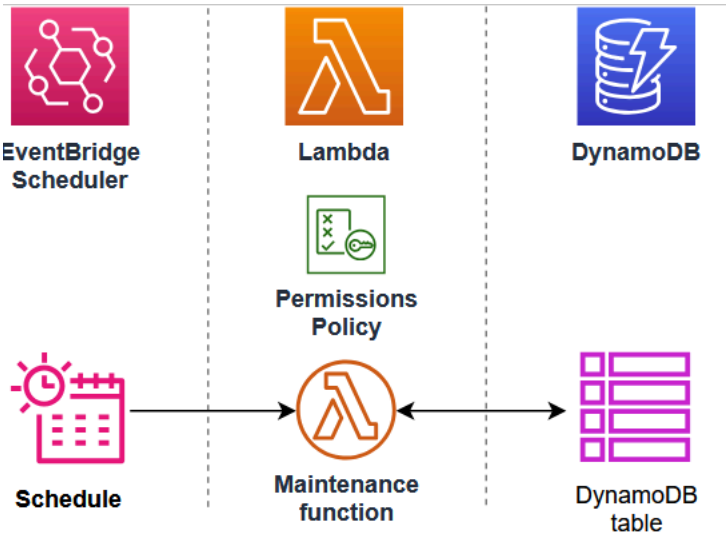
Saat menerapkan sumber daya Anda AWS SAM, Anda perlu mengambil beberapa langkah ekstra untuk menyertakan distribusi sumber yang tepat dalam paket penerapan Anda. Karena AWS SAM tidak akan menginstal dependensi untuk platform yang berbeda dari mesin build Anda, menentukan distribusi sumber (`.whl`file) yang benar dalam `requirements.txt` file Anda tidak akan berfungsi jika mesin build Anda menggunakan sistem operasi atau arsitektur yang berbeda dari lingkungan eksekusi Lambda. Sebagai gantinya, Anda harus melakukan salah satu hal berikut:

- Gunakan `--use-container` opsi saat menjalankan`sam build`. Saat Anda menentukan opsi ini, AWS SAM unduh gambar dasar kontainer yang kompatibel dengan lingkungan eksekusi Lambda dan buat paket penerapan fungsi Anda dalam wadah Docker menggunakan gambar itu. Untuk mempelajari lebih lanjut, lihat Membangun fungsi Lambda di dalam wadah yang disediakan.
- Buat sendiri paket.zip deployment fungsi Anda menggunakan biner distribusi sumber yang benar dan simpan file.zip di direktori yang Anda tentukan seperti `CodeUri` di template. AWS SAM Untuk mempelajari lebih lanjut tentang membangun paket deployment .zip untuk Python menggunakan distribusi biner, lihat dan. the section called “Membuat paket penerapan.zip dengan dependensi” the section called “Membuat paket penerapan.zip dengan pustaka asli”

Membuat aplikasi untuk melakukan pemeliharaan database terjadwal

Anda dapat menggunakan AWS Lambda untuk mengganti proses terjadwal seperti pencadangan sistem otomatis, konversi file, dan tugas pemeliharaan. Dalam contoh ini, Anda membuat aplikasi tanpa server yang melakukan pemeliharaan terjadwal rutin pada tabel DynamoDB dengan menghapus entri lama. Aplikasi ini menggunakan EventBridge Scheduler untuk menjalankan fungsi Lambda pada jadwal cron. Saat dipanggil, fungsi menanyakan tabel untuk item yang lebih tua dari satu tahun, dan menghapusnya. Fungsi mencatat setiap item yang dihapus di CloudWatch Log.

Untuk mengimplementasikan contoh ini, pertama buat tabel DynamoDB dan isi dengan beberapa data uji untuk fungsi Anda untuk query. Kemudian, buat fungsi Lambda Python dengan pemicu EventBridge Scheduler dan peran eksekusi IAM yang memberikan izin fungsi untuk membaca, dan menghapus, item dari tabel Anda.



> **Tip**
>
> Jika Anda baru mengenal Lambda, kami sarankan Anda menyelesaikan tutorial the section called “Buat fungsi pertama Anda” sebelum membuat aplikasi contoh ini.

Anda dapat menerapkan aplikasi secara manual dengan membuat dan mengonfigurasi sumber daya dengan aplikasi. Konsol Manajemen AWS Anda juga dapat menerapkan aplikasi dengan menggunakan AWS Serverless Application Model (AWS SAM). AWS SAM adalah alat infrastruktur sebagai kode (IAc). Dengan IAc, Anda tidak membuat sumber daya secara manual, tetapi mendefinisikannya dalam kode dan kemudian menerapkannya secara otomatis.

Jika Anda ingin mempelajari lebih lanjut tentang menggunakan Lambda dengan IAc sebelum menerapkan aplikasi contoh ini, lihat. the section called “Infrastruktur sebagai kode (IAc)”

Prasyarat

Sebelum Anda dapat membuat aplikasi contoh, pastikan Anda memiliki alat dan program baris perintah yang diperlukan diinstal.

- Python

  Untuk mengisi tabel DynamoDB yang Anda buat untuk menguji aplikasi, contoh ini menggunakan skrip Python dan file CSV untuk menulis data ke dalam tabel. Pastikan Anda memiliki Python versi 3.8 atau yang lebih baru diinstal pada mesin Anda.
- AWS SAM CLI

Jika Anda ingin membuat tabel DynamoDB dan menerapkan aplikasi contoh AWS SAM menggunakan, Anda perlu menginstal CLI. AWS SAM Ikuti petunjuk penginstalan di Panduan AWS SAM Pengguna.

- AWS CLI

  Untuk menggunakan skrip Python yang disediakan untuk mengisi tabel pengujian Anda, Anda harus menginstal dan mengkonfigurasi file. AWS CLI Ini karena skrip menggunakan AWS SDK untuk Python (Boto3), yang membutuhkan akses ke kredensi AWS Identity and Access Management (IAM) Anda. Anda juga perlu AWS CLI diinstal untuk menyebarkan sumber daya menggunakan AWS SAM. Instal CLI dengan mengikuti petunjuk penginstalan di AWS Command Line Interface Panduan Pengguna.

- Docker

  Untuk menerapkan aplikasi menggunakan AWS SAM, Docker juga harus diinstal pada mesin build Anda. Ikuti petunjuk di Instal Docker Engine di situs web dokumentasi Docker.

Mengunduh contoh file aplikasi

Untuk membuat database contoh dan aplikasi pemeliharaan terjadwal, Anda perlu membuat file berikut di direktori proyek Anda:

Contoh file database

- `template.yaml`- AWS SAM Template yang dapat Anda gunakan untuk membuat tabel DynamoDB
- `sample_data.csv`- file CSV yang berisi data sampel untuk dimuat ke dalam tabel Anda
- `load_sample_data.py`- skrip Python yang menulis data dalam file CSV ke dalam tabel

Scheduled-maintenance file aplikasi

- `lambda_function.py`- kode fungsi Python untuk fungsi Lambda yang melakukan pemeliharaan database
- `requirements.txt`- file manifes yang mendefinisikan dependensi yang dibutuhkan kode fungsi Python Anda
- `template.yaml`- AWS SAM Template yang dapat Anda gunakan untuk menyebarkan aplikasi

File uji

- `test_app.py`- skrip Python yang memindai tabel dan mengonfirmasi keberhasilan operasi fungsi Anda dengan mengeluarkan semua catatan yang lebih tua dari satu tahun

Perluas bagian berikut untuk melihat kode dan mempelajari lebih lanjut tentang peran setiap file dalam membuat dan menguji aplikasi Anda. Untuk membuat file di mesin lokal Anda, salin dan tempel kode di bawah ini.

AWS SAM template (contoh tabel DynamoDB)

Salin dan tempel kode berikut ke dalam file bernama`template.yaml`.

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: SAM Template for DynamoDB Table with Order_number as Partition Key and
 Date as Sort Key

Resources:
  MyDynamoDBTable:
    Type: AWS::DynamoDB::Table
    DeletionPolicy: Retain
    UpdateReplacePolicy: Retain
    Properties:
      TableName: MyOrderTable
      BillingMode: PAY_PER_REQUEST
      AttributeDefinitions:
        - AttributeName: Order_number
          AttributeType: S
        - AttributeName: Date
          AttributeType: S
      KeySchema:
        - AttributeName: Order_number
          KeyType: HASH
        - AttributeName: Date
          KeyType: RANGE
      SSESpecification:
        SSEEnabled: true
      GlobalSecondaryIndexes:
        - IndexName: Date-index
          KeySchema:
            - AttributeName: Date
```

```
              KeyType: HASH
          Projection:
            ProjectionType: ALL
      PointInTimeRecoverySpecification:
        PointInTimeRecoveryEnabled: true

Outputs:
  TableName:
    Description: DynamoDB Table Name
    Value: !Ref MyDynamoDBTable
  TableArn:
    Description: DynamoDB Table ARN
    Value: !GetAtt MyDynamoDBTable.Arn
```

> **Note**
>
> AWS SAM template menggunakan konvensi penamaan standar`template.yaml`. Dalam contoh ini, Anda memiliki dua file template - satu untuk membuat database contoh dan satu lagi untuk membuat aplikasi itu sendiri. Simpan di sub-direktori terpisah di folder proyek Anda.

AWS SAM Template ini mendefinisikan resource tabel DynamoDB yang Anda buat untuk menguji aplikasi Anda. Tabel menggunakan kunci utama `Order_number` dengan semacam kunci`Date`. Agar fungsi Lambda Anda menemukan item secara langsung berdasarkan tanggal, kami juga mendefinisikan Indeks Sekunder Global bernama. `Date-index`

Untuk mempelajari selengkapnya tentang membuat dan mengonfigurasi tabel DynamoDB menggunakan *`AWS::DynamoDB::Table`* sumber daya, AWS lihat: :DynamoDB: :Table di Panduan Pengguna.AWS CloudFormation

Contoh file data database

Salin dan tempel kode berikut ke dalam file bernama`sample_data.csv`.

```
Date,Order_number,CustomerName,ProductID,Quantity,TotalAmount
2023-09-01,ORD001,Alejandro Rosalez,PROD123,2,199.98
2023-09-01,ORD002,Akua Mansa,PROD456,1,49.99
2023-09-02,ORD003,Ana Carolina Silva,PROD789,3,149.97
2023-09-03,ORD004,Arnav Desai,PROD123,1,99.99
2023-10-01,ORD005,Carlos Salazar,PROD456,2,99.98
2023-10-02,ORD006,Diego Ramirez,PROD789,1,49.99
2023-10-03,ORD007,Efua Owusu,PROD123,4,399.96
```

```
2023-10-04,ORD008,John Stiles,PROD456,2,99.98
2023-10-05,ORD009,Jorge Souza,PROD789,3,149.97
2023-10-06,ORD010,Kwaku Mensah,PROD123,1,99.99
2023-11-01,ORD011,Li Juan,PROD456,5,249.95
2023-11-02,ORD012,Marcia Oliveria,PROD789,2,99.98
2023-11-03,ORD013,Maria Garcia,PROD123,3,299.97
2023-11-04,ORD014,Martha Rivera,PROD456,1,49.99
2023-11-05,ORD015,Mary Major,PROD789,4,199.96
2023-12-01,ORD016,Mateo Jackson,PROD123,2,199.99
2023-12-02,ORD017,Nikki Wolf,PROD456,3,149.97
2023-12-03,ORD018,Pat Candella,PROD789,1,49.99
2023-12-04,ORD019,Paulo Santos,PROD123,5,499.95
2023-12-05,ORD020,Richard Roe,PROD456,2,99.98
2024-01-01,ORD021,Saanvi Sarkar,PROD789,3,149.97
2024-01-02,ORD022,Shirley Rodriguez,PROD123,1,99.99
2024-01-03,ORD023,Sofia Martinez,PROD456,4,199.96
2024-01-04,ORD024,Terry Whitlock,PROD789,2,99.98
2024-01-05,ORD025,Wang Xiulan,PROD123,3,299.97
```

File ini berisi beberapa contoh data uji untuk mengisi tabel DynamoDB Anda dengan format nilai dipisahkan koma (CSV) standar.

Skrip Python untuk memuat data sampel

Salin dan tempel kode berikut ke dalam file bernama`load_sample_data.py`.

```
import boto3
import csv
from decimal import Decimal

# Initialize the DynamoDB client
dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('MyOrderTable')
print("DDB client initialized.")

def load_data_from_csv(filename):
    with open(filename, 'r') as file:
        csv_reader = csv.DictReader(file)
        for row in csv_reader:
            item = {
                'Order_number': row['Order_number'],
                'Date': row['Date'],
                'CustomerName': row['CustomerName'],
```

```
                'ProductID': row['ProductID'],
                'Quantity': int(row['Quantity']),
                'TotalAmount': Decimal(str(row['TotalAmount']))
            }
            table.put_item(Item=item)
            print(f"Added item: {item['Order_number']} - {item['Date']}")

if __name__ == "__main__":
    load_data_from_csv('sample_data.csv')
    print("Data loading completed.")
```

Script Python ini pertama kali menggunakan AWS SDK untuk Python (Boto3) untuk membuat koneksi ke tabel DynamoDB Anda. Kemudian iterasi di setiap baris dalam file CSV contoh-data, membuat item dari baris itu, dan menulis item ke tabel DynamoDB menggunakan boto3 SDK.

Kode fungsi Python

Salin dan tempel kode berikut ke dalam file bernama`lambda_function.py`.

```
import boto3
from datetime import datetime, timedelta
from boto3.dynamodb.conditions import Key, Attr
import logging

logger = logging.getLogger()
logger.setLevel("INFO")

def lambda_handler(event, context):
    # Initialize the DynamoDB client
    dynamodb = boto3.resource('dynamodb')

    # Specify the table name
    table_name = 'MyOrderTable'
    table = dynamodb.Table(table_name)

    # Get today's date
    today = datetime.now()

    # Calculate the date one year ago
    one_year_ago = (today - timedelta(days=365)).strftime('%Y-%m-%d')

    # Scan the table using a global secondary index
    response = table.scan(
```

```
        IndexName='Date-index',
        FilterExpression='#date < :one_year_ago',
        ExpressionAttributeNames={
            '#date': 'Date'
        },
        ExpressionAttributeValues={
            ':one_year_ago': one_year_ago
        }
    )

     # Delete old items
    with table.batch_writer() as batch:
        for item in response['Items']:
            Order_number = item['Order_number']
            batch.delete_item(
                Key={
                    'Order_number': Order_number,
                    'Date': item['Date']
                }
            )
            logger.info(f'deleted order number {Order_number}')

    # Check if there are more items to scan
    while 'LastEvaluatedKey' in response:
        response = table.scan(
            IndexName='DateIndex',
            FilterExpression='#date < :one_year_ago',
            ExpressionAttributeNames={
                '#date': 'Date'
            },
            ExpressionAttributeValues={
                ':one_year_ago': one_year_ago
            },
            ExclusiveStartKey=response['LastEvaluatedKey']
        )

        # Delete old items
        with table.batch_writer() as batch:
            for item in response['Items']:
                batch.delete_item(
                    Key={
                        'Order_number': item['Order_number'],
                        'Date': item['Date']
                    }
```

```
                )

    return {
        'statusCode': 200,
        'body': 'Cleanup completed successfully'
    }
```

Kode fungsi Python berisi fungsi handler () `lambda_handler` yang dijalankan Lambda saat fungsi Anda dipanggil.

Ketika fungsi dipanggil oleh EventBridge Scheduler, ia menggunakan AWS SDK untuk Python (Boto3) untuk membuat koneksi ke tabel DynamoDB di mana tugas pemeliharaan terjadwal akan dilakukan. Kemudian menggunakan `datetime` pustaka Python untuk menghitung tanggal satu tahun yang lalu, sebelum memindai tabel untuk item yang lebih tua dari ini dan menghapusnya.

Perhatikan bahwa tanggapan dari kueri DynamoDB dan operasi pemindaian dibatasi hingga ukuran maksimum 1 MB. Jika respon lebih besar dari 1 MB, DynamoDB paginates data dan mengembalikan `LastEvaluatedKey` elemen dalam respon. Untuk memastikan bahwa fungsi kami memproses semua catatan dalam tabel, kami memeriksa keberadaan kunci ini dan terus melakukan pemindaian tabel dari posisi yang dievaluasi terakhir hingga seluruh tabel dipindai.

`file manifes requirements.txt`

Salin dan tempel kode berikut ke dalam file bernama`requirements.txt`.

```
boto3
```

Untuk contoh ini, kode fungsi Anda hanya memiliki satu dependensi yang bukan bagian dari pustaka Python standar - SDK for Python (Boto3) yang digunakan fungsi untuk memindai dan menghapus item dari tabel DynamoDB.

> **Note**
>
> Versi SDK untuk Python (Boto3) disertakan sebagai bagian dari runtime Lambda, sehingga kode Anda akan berjalan tanpa menambahkan Boto3 ke paket penerapan fungsi Anda. Namun, untuk mempertahankan kontrol penuh dependensi fungsi Anda dan menghindari kemungkinan masalah dengan misalignment versi, praktik terbaik untuk Python adalah menyertakan semua dependensi fungsi dalam paket penerapan fungsi Anda. Lihat the section called “Dependensi runtime dengan Python” untuk mempelajari selengkapnya.

AWS SAM template (aplikasi pemeliharaan terjadwal)

Salin dan tempel kode berikut ke dalam file bernama`template.yaml`.

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: SAM Template for Lambda function and EventBridge Scheduler rule

Resources:
  MyLambdaFunction:
    Type: AWS::Serverless::Function
    Properties:
      FunctionName: ScheduledDBMaintenance
      CodeUri: ./
      Handler: lambda_function.lambda_handler
      Runtime: python3.11
      Architectures:
        - x86_64
      Events:
        ScheduleEvent:
          Type: ScheduleV2
          Properties:
            ScheduleExpression: cron(0 3 1 * ? *)
            Description: Run on the first day of every month at 03:00 AM
      Policies:
        - CloudWatchLogsFullAccess
        - Statement:
            - Effect: Allow
              Action:
                - dynamodb:Scan
                - dynamodb:BatchWriteItem
              Resource: !Sub 'arn:aws:dynamodb:${AWS::Region}:${AWS::AccountId}:table/
MyOrderTable'

  LambdaLogGroup:
    Type: AWS::Logs::LogGroup
    Properties:
      LogGroupName: !Sub /aws/lambda/${MyLambdaFunction}
      RetentionInDays: 30

Outputs:
  LambdaFunctionName:
    Description: Lambda Function Name
    Value: !Ref MyLambdaFunction
```

```
  LambdaFunctionArn:
    Description: Lambda Function ARN
    Value: !GetAtt MyLambdaFunction.Arn
```

> **Note**
>
> AWS SAM template menggunakan konvensi penamaan standar`template.yaml`. Dalam contoh ini, Anda memiliki dua file template - satu untuk membuat database contoh dan satu lagi untuk membuat aplikasi itu sendiri. Simpan di sub-direktori terpisah di folder proyek Anda.

AWS SAM Template ini mendefinisikan resource untuk aplikasi Anda. Kami mendefinisikan fungsi Lambda menggunakan sumber daya. `AWS::Serverless::Function` Jadwal EventBridge Scheduler dan pemicu untuk memanggil fungsi Lambda dibuat dengan menggunakan `Events` properti sumber daya ini menggunakan tipe. `ScheduleV2` Untuk mempelajari selengkapnya tentang menentukan jadwal EventBridge Penjadwal dalam AWS SAM templat, lihat ScheduleV2 di Panduan Pengembang.AWS Serverless Application Model

Selain fungsi Lambda dan jadwal EventBridge Scheduler, kami juga mendefinisikan grup CloudWatch log untuk fungsi Anda untuk mengirim catatan item yang dihapus ke.

Skrip uji

Salin dan tempel kode berikut ke dalam file bernama`test_app.py`.

```
import boto3
from datetime import datetime, timedelta
import json

# Initialize the DynamoDB client
dynamodb = boto3.resource('dynamodb')

# Specify your table name
table_name = 'YourTableName'
table = dynamodb.Table(table_name)

# Get the current date
current_date = datetime.now()

# Calculate the date one year ago
one_year_ago = current_date - timedelta(days=365)
```

```
# Convert the date to string format (assuming the date in DynamoDB is stored as a
 string)
one_year_ago_str = one_year_ago.strftime('%Y-%m-%d')

# Scan the table
response = table.scan(
    FilterExpression='#date < :one_year_ago',
    ExpressionAttributeNames={
        '#date': 'Date'
    },
    ExpressionAttributeValues={
        ':one_year_ago': one_year_ago_str
    }
)

# Process the results
old_records = response['Items']

# Continue scanning if we have more items (pagination)
while 'LastEvaluatedKey' in response:
    response = table.scan(
        FilterExpression='#date < :one_year_ago',
        ExpressionAttributeNames={
            '#date': 'Date'
        },
        ExpressionAttributeValues={
            ':one_year_ago': one_year_ago_str
        },
        ExclusiveStartKey=response['LastEvaluatedKey']
    )
    old_records.extend(response['Items'])

for record in old_records:
    print(json.dumps(record))

# The total number of old records should be zero.
print(f"Total number of old records: {len(old_records)}")
```

Skrip pengujian ini menggunakan AWS SDK untuk Python (Boto3) untuk membuat koneksi ke tabel DynamoDB Anda dan memindai item yang lebih tua dari satu tahun. Untuk mengonfirmasi apakah fungsi Lambda telah berjalan dengan sukses, pada akhir pengujian, fungsi mencetak jumlah catatan

yang lebih tua dari satu tahun masih dalam tabel. Jika fungsi Lambda berhasil, jumlah catatan lama dalam tabel harus nol.

Membuat dan mengisi tabel contoh DynamoDB

Untuk menguji aplikasi pemeliharaan terjadwal, pertama-tama Anda membuat tabel DynamoDB dan mengisinya dengan beberapa data sampel. Anda dapat membuat tabel baik secara manual menggunakan Konsol Manajemen AWS atau dengan menggunakan AWS SAM. Kami menyarankan Anda menggunakan AWS SAM untuk membuat dan mengkonfigurasi tabel dengan cepat menggunakan beberapa AWS CLI perintah.

Console

Untuk membuat tabel DynamoDB

1. Buka halaman Tabel di konsol DynamoDB.
2. Pilih Buat tabel.
3. Buat tabel dengan melakukan hal berikut:
   a. Di bawah Rincian tabel, untuk nama Tabel, masukkan**`MyOrderTable`**.
   b. Untuk kunci Partition, masukkan **`Order_number`** dan biarkan tipe sebagai String.
   c. Untuk tombol Sort, masukkan **`Date`** dan biarkan tipe sebagai String.
   d. Biarkan pengaturan Tabel diatur ke Pengaturan default dan pilih Buat tabel.
4. Ketika tabel Anda telah selesai dibuat dan Statusnya ditampilkan sebagai Aktif, buat indeks sekunder global (GSI) dengan melakukan hal berikut. Aplikasi Anda menggunakan GSI ini untuk mencari item secara langsung berdasarkan tanggal untuk menentukan apa yang akan dihapus.
   a. Pilih MyOrderTabledari daftar tabel.
   b. Pilih tab Indeks.
   c. Di bawah Indeks sekunder global, pilih Buat indeks.
   d. Di bawah Detail indeks, masukkan **`Date`** untuk kunci Partition dan biarkan tipe Data diatur ke String.
   e. Untuk Nama indeks, masukkan **`Date-index`**.
   f. Biarkan semua parameter lain diatur ke nilai defaultnya, gulir ke bagian bawah halaman, dan pilih Buat indeks.

AWS SAM

Untuk membuat tabel DynamoDB

1. Arahkan ke folder tempat Anda menyimpan `template.yaml` file untuk tabel DynamoDB. Perhatikan bahwa contoh ini menggunakan dua `template.yaml` file. Pastikan mereka disimpan dalam sub-folder terpisah dan bahwa Anda berada di folder yang benar berisi template untuk membuat tabel DynamoDB Anda.
2. Jalankan perintah berikut.

   ```
   sam build
   ```

   Perintah ini mengumpulkan artefak build untuk sumber daya yang ingin Anda terapkan dan menempatkannya dalam format dan lokasi yang tepat untuk menerapkannya.
3. Untuk membuat sumber daya DynamoDB yang ditentukan dalam `template.yaml` file, jalankan perintah berikut.

   ```
   sam deploy --guided
   ```

   Menggunakan `--guided` tanda berarti AWS SAM menunjukkan kepada Anda petunjuk untuk memandu Anda melalui proses penerapan. Untuk penyebaran ini, masukkan a `Stack name` dari**`cron-app-test-db`**, dan terima default untuk semua opsi lain dengan menggunakan Enter.

   Setelah AWS SAM selesai membuat sumber daya DynamoDB, Anda akan melihat pesan berikut.

   ```
   Successfully created/updated stack - cron-app-test-db in us-west-2
   ```
4. Anda juga dapat mengonfirmasi bahwa tabel DynamoDB telah dibuat dengan membuka halaman Tabel dari konsol DynamoDB. Anda harus melihat tabel bernama`MyOrderTable`.

Setelah membuat tabel, Anda selanjutnya menambahkan beberapa data sampel untuk menguji aplikasi Anda. File CSV yang `sample_data.csv` Anda unduh sebelumnya berisi sejumlah contoh entri yang terdiri dari nomor pesanan, tanggal, dan informasi pelanggan dan pesanan. Gunakan skrip python yang disediakan `load_sample_data.py` untuk menambahkan data ini ke tabel Anda.

Untuk menambahkan data sampel ke tabel

1. Arahkan ke direktori yang berisi `load_sample_data.py` file `sample_data.csv` dan. Jika file-file ini berada di direktori terpisah, pindahkan agar file tersebut disimpan di lokasi yang sama.
2. Buat lingkungan virtual Python untuk menjalankan skrip dengan menjalankan perintah berikut. Kami menyarankan Anda menggunakan lingkungan virtual karena pada langkah berikut Anda harus menginstal AWS SDK untuk Python (Boto3).

   ```
   python -m venv venv
   ```

3. Aktifkan lingkungan virtual dengan menjalankan perintah berikut.

   ```
   source venv/bin/activate
   ```

4. Instal SDK for Python (Boto3) di lingkungan virtual Anda dengan menjalankan perintah berikut. Skrip menggunakan pustaka ini untuk terhubung ke tabel DynamoDB Anda dan menambahkan item.

   ```
   pip install boto3
   ```

5. Jalankan skrip untuk mengisi tabel dengan menjalankan perintah berikut.

   ```
   python load_sample_data.py
   ```

   Jika skrip berjalan dengan sukses, skrip harus mencetak setiap item ke konsol saat memuat dan melaporkannya`Data loading completed`.
6. Nonaktifkan lingkungan virtual dengan menjalankan perintah berikut.

   ```
   deactivate
   ```

7. Anda dapat memverifikasi bahwa data telah dimuat ke tabel DynamoDB Anda dengan melakukan hal berikut:

   a. Buka halaman Jelajahi item dari konsol DynamoDB dan pilih tabel Anda (). `MyOrderTable`

   b. Di panel Item yang dikembalikan, Anda akan melihat 25 item dari file CSV yang ditambahkan skrip ke tabel.

Membuat aplikasi pemeliharaan terjadwal

Anda dapat membuat dan menerapkan sumber daya untuk aplikasi contoh ini langkah demi langkah menggunakan Konsol Manajemen AWS atau dengan menggunakan AWS SAM. Dalam lingkungan produksi, kami menyarankan Anda menggunakan alat Infrustracture-as-Code (IAc) seperti AWS SAM menyebarkan aplikasi tanpa server secara berulang tanpa menggunakan proses manual.

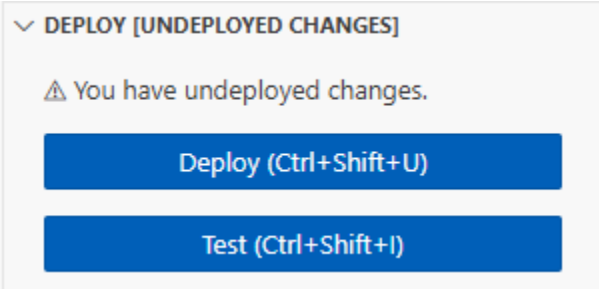
Untuk contoh ini, ikuti petunjuk konsol untuk mempelajari cara mengonfigurasi setiap AWS sumber daya secara terpisah, atau ikuti AWS SAM petunjuk untuk menerapkan aplikasi dengan cepat menggunakan AWS CLI perintah.

Console

Untuk membuat fungsi menggunakan Konsol Manajemen AWS

Pertama, buat fungsi yang berisi kode starter dasar. Anda kemudian mengganti kode ini dengan kode fungsi Anda sendiri dengan menyalin dan menempelkan kode langsung di editor kode Lambda, atau dengan mengunggah kode Anda sebagai paket. `.zip` Untuk tugas ini, kami sarankan menyalin dan menempelkan kode.

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih Buat Fungsi Acara.
3. Pilih Tulis dari awal.
4. Di bagian Informasi dasar, lakukan hal berikut:
   a. Untuk Nama fungsi, masukkan **`ScheduledDBMaintenance`**.
   b. Untuk Runtime pilih versi Python terbaru.
   c. Untuk Arsitektur, pilih x86_64.
5. Pilih Buat Fungsi Acara.
6. Setelah fungsi Anda dibuat, Anda dapat mengonfigurasi fungsi Anda dengan kode fungsi yang disediakan.
   a. Di panel sumber Kode, ganti kode Hello world yang dibuat Lambda dengan kode fungsi Python dari file yang Anda simpan `lambda_function.py` sebelumnya.
   b. Di bagian DEPLOY, pilih Deploy untuk memperbarui kode fungsi Anda:



Untuk mengkonfigurasi memori fungsi dan batas waktu (konsol)

1. Pilih tab Konfigurasi untuk fungsi Anda.
2. Di panel konfigurasi Umum, pilih Edit.
3. Atur Memori ke 256 MB dan Timeout menjadi 15 detik. Jika Anda memproses tabel besar dengan banyak catatan, misalnya dalam kasus lingkungan produksi, Anda dapat mempertimbangkan untuk mengatur Timeout ke jumlah yang lebih besar. Ini memberi fungsi Anda lebih banyak waktu untuk memindai, dan membersihkan database.
4. Pilih Simpan.

Untuk mengkonfigurasi format log (konsol)

Anda dapat mengonfigurasi fungsi Lambda untuk mengeluarkan log dalam teks tidak terstruktur atau format JSON. Kami menyarankan Anda menggunakan format JSON untuk log agar lebih mudah untuk mencari dan memfilter data log. Untuk mempelajari lebih lanjut tentang opsi konfigurasi log Lambda, lihat. the section called “Mengkonfigurasi kontrol logging lanjutan untuk Lambda Event Functions”

1. Pilih tab Konfigurasi untuk fungsi Anda.
2. Pilih Alat pemantauan dan operasi.
3. Di panel konfigurasi Logging, pilih Edit.
4. Untuk konfigurasi Logging, pilih JSON.
5. Pilih Simpan.

Untuk mengatur izin IAM

Untuk memberikan fungsi Anda izin yang diperlukan untuk membaca dan menghapus item DynamoDB, Anda perlu menambahkan kebijakan ke peran eksekusi fungsi Anda yang menentukan izin yang diperlukan.

1. Buka tab Konfigurasi, lalu pilih Izin dari bilah navigasi kiri.
2. Pilih nama peran di bawah Peran eksekusi.
3. Di konsol IAM, pilih Tambahkan izin, lalu Buat kebijakan sebaris.
4. Gunakan editor JSON dan masukkan kebijakan berikut:

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:Scan",
                "dynamodb:DeleteItem",
                "dynamodb:BatchWriteItem"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/MyOrderTable"
        }
    ]
}
```

5. Beri nama kebijakan**`DynamoDBCleanupPolicy`**, lalu buat.

Untuk mengatur EventBridge Scheduler sebagai pemicu (konsol)

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih fungsi Anda (`ScheduledDBMaintenance`).
3. Di panel Ikhtisar fungsi, pilih Tambah pemicu.
4. Dari dropdown sumber pemicu, pilih Scheduler.
5. Konfigurasikan jadwal dengan melakukan hal berikut:
   a. Untuk nama Jadwal, masukkan**`DynamoDBCleanupSchedule`**.
   b. Untuk grup Jadwal, tinggalkan default (`default`).

c. Untuk jenis Jadwal, pilih Jadwal berulang dan masukkan ekspresi cron berikut:. **`cron(0 3 1 * ? *)`**

   Saat dievaluasi, ekspresi cron ini berjalan pada hari pertama setiap bulan pukul 03:00 pagi.

d. Untuk jendela waktu Fleksibel, pilih Mati.

e. Biarkan bidang Payload kosong.

f. Untuk peran eksekusi, pilih Buat peran baru untuk jadwal ini dan masukkan nama peran. EventBridge Scheduler membuat kebijakan baru dengan izin yang diperlukan jadwal untuk menjalankan fungsi Anda.

6. Pilih Tambahkan.

AWS SAM

Untuk menerapkan aplikasi menggunakan AWS SAM

1. Arahkan ke folder tempat Anda menyimpan `template.yaml` file untuk aplikasi. Perhatikan bahwa contoh ini menggunakan dua `template.yaml` file. Pastikan mereka disimpan dalam sub-folder terpisah dan bahwa Anda berada di folder yang benar berisi template untuk membuat aplikasi.
2. Salin `requirements.txt` file `lambda_function.py` dan file yang Anda unduh sebelumnya ke folder yang sama. Lokasi kode yang ditentukan dalam AWS SAM template adalah `./`, yang berarti lokasi saat ini. AWS SAM mencari di folder ini untuk kode fungsi Lambda saat Anda mencoba menerapkan aplikasi.
3. Jalankan perintah berikut.

   ```
   sam build --use-container
   ```

   Perintah ini mengumpulkan artefak build untuk sumber daya yang ingin Anda terapkan dan menempatkannya dalam format dan lokasi yang tepat untuk menerapkannya. Menentukan `--use-container` opsi membangun fungsi Anda di dalam wadah Lambda-like Docker. Kami menggunakannya di sini sehingga Anda tidak perlu menginstal Python 3.12 di mesin lokal Anda agar build berfungsi.
4. Untuk membuat sumber daya Lambda dan EventBridge Scheduler yang ditentukan dalam `template.yaml` file, jalankan perintah berikut.

```
sam deploy --guided
```

Menggunakan `--guided` tanda berarti AWS SAM menunjukkan kepada Anda petunjuk untuk memandu Anda melalui proses penerapan. Untuk penyebaran ini, masukkan a `Stack name` dari**`cron-maintenance-app`**, dan terima default untuk semua opsi lain dengan menggunakan Enter.

Setelah AWS SAM selesai membuat sumber daya Lambda dan EventBridge Scheduler, Anda akan melihat pesan berikut.

```
Successfully created/updated stack - cron-maintenance-app in us-west-2
```

5. Anda juga dapat mengonfirmasi bahwa fungsi Lambda telah dibuat dengan membuka halaman Fungsi konsol Lambda. Anda akan melihat fungsi bernama`ScheduledDBMaintenance`.

Menguji aplikasi

Untuk menguji apakah jadwal Anda memicu fungsi Anda dengan benar, dan bahwa fungsi Anda membersihkan catatan dengan benar dari database, Anda dapat mengubah jadwal sementara untuk dijalankan sekali pada waktu tertentu. Anda kemudian dapat menjalankan `sam deploy` lagi untuk mengatur ulang jadwal pengulangan Anda agar berjalan sebulan sekali.

Untuk menjalankan aplikasi menggunakan Konsol Manajemen AWS

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih fungsi Anda (`ScheduledDBMaintenance`).
3. Di bagian Pemicu, pilih `DynamoDBCleanupSchedule` pemicu.
4. Pilih Edit.
5. Untuk jenis Jadwal, pilih One-time jadwal.
6. Setel waktu pemanggilan Anda ke beberapa menit dari sekarang, lalu pilih Simpan.

Setelah jadwal berjalan dan memanggil targetnya, Anda menjalankan `test_app.py` skrip untuk memverifikasi bahwa fungsi Anda berhasil menghapus semua catatan lama dari tabel DynamoDB.

Untuk memverifikasi bahwa catatan lama dihapus menggunakan skrip Python

1. Di baris perintah Anda, arahkan ke folder tempat Anda menyimpan`test_app.py`.
2. Jalankan penulisan.

   ```
   python test_app.py
   ```

   Jika berhasil, Anda melihat output berikut.

   ```
   Total number of old records: 0
   ```

Langkah selanjutnya

Anda sekarang dapat memodifikasi EventBridge jadwal Scheduler untuk memenuhi persyaratan aplikasi khusus Anda. EventBridge Scheduler mendukung ekspresi jadwal berikut: cron, rate, dan jadwal satu kali.

Untuk informasi selengkapnya tentang Ekspresi EventBridge jadwal penjadwal, lihat Jenis jadwal di Panduan Pengguna EventBridge Penjadwal. Manajemen Akses di Panduan Pengguna IAM

Bahasa pemrograman

Lambda Event Functions (sebelumnya dikenal sebagai fungsi Lambda) mendukung berbagai bahasa pemrograman populer. Masing-masing memiliki pengalaman pemrograman idiomatik yang disesuaikan dengan bahasa itu.

Setiap bahasa pemrograman didukung oleh runtime yang sesuai, yang menyediakan lingkungan eksekusi khusus bahasa untuk fungsi Anda.

- the section called “Membangun dengan Node.js”
- the section called “Membangun dengan TypeScript”
- the section called “Membangun dengan Python”
- the section called “Membangun dengan Ruby”
- the section called “Membangun dengan Java”
- the section called “Membangun dengan Go”

- the section called “Membangun dengan C #”
- the section called “Membangun dengan PowerShell”
- the section called “Membangun dengan Karat”

Membangun Fungsi Acara Lambda dengan Node.js

Anda bisa lari JavaScript kode dengan Node.js di AWS Lambda Fungsi Acara. Lambda menyediakan waktu proses untuk Node.js yang menjalankan kode Anda untuk memproses peristiwa. Kode Anda berjalan di lingkungan yang mencakup AWS SDK untuk JavaScript, dengan kredensi dari AWS Identity and Access Management (IAM) peran yang Anda kelola. Untuk mempelajari lebih lanjut tentang versi SDK yang disertakan dengan Node.js runtime, lihat the section called “Runtime-included Versi SDK”.

Lambda mendukung yang berikut Node.js waktu proses.

Nama	Pengidentifikasi	Sistem operasi	Tanggal pengusangan	Blokir fungsi buat	Pembaruan fungsi blok
Node.js 26	`nodejs26.x`	Amazon Linux 2023	Tidak dijadwalkan	Tidak dijadwalkan	Tidak dijadwalkan
Node.js 24	`nodejs24.x`	Amazon Linux 2023	30 April 2028	1 Juni 2028	1 Juli 2028
Node.js 22	`nodejs22.x`	Amazon Linux 2023	30 April 2027	29 Juli 2027	31 Agu 2027

Untuk membuat Node.js Fungsi Acara

1. Buka Konsol Lambda.
2. Pilih Buat Fungsi Acara.
3. Konfigurasikan pengaturan berikut:
   - Nama fungsi: Masukkan nama untuk fungsi.
   - Waktu pelaksanaan: Pilih Node.js 24.x.
4. Memilih Buat Fungsi Acara.

Konsol membuat Lambda Event Function dengan satu file sumber bernama `index.mjs`. Anda dapat mengedit file ini dan menambahkan lebih banyak file di editor kode bawaan. Di MENYEBARKAN bagian, pilih Menyebarkan untuk memperbarui kode fungsi Anda. Kemudian, untuk menjalankan kode Anda, pilih Buat acara pengujian di ACARA UJI bagian.

File `index.mjs` mengekspor fungsi bernama `handler` yang menerima objek peristiwa dan objek konteks. Ini adalah fungsi handler yang dipanggil Lambda saat fungsi tersebut dipanggil. The Node.js runtime fungsi mendapatkan peristiwa pemanggilan dari Lambda dan meneruskannya ke handler. Dalam konfigurasi fungsi, nilai handler adalah `index.handler`.

Saat Anda menyimpan kode fungsi, konsol Lambda membuat paket penyebaran arsip file.zip. Ketika Anda mengembangkan kode fungsi Anda di luar konsol (menggunakan IDE), Anda perlu membuat paket penerapan untuk mengunggah kode Anda ke fungsi Lambda.

Runtime fungsi meneruskan objek konteks ke handler, selain peristiwa invokasi. Objek konteks berisi informasi tambahan tentang lingkungan invokasi, fungsi, dan eksekusi. Informasi selengkapnya tersedia dari variabel lingkungan.

Fungsi Lambda Anda dilengkapi dengan CloudWatch Grup log log. Runtime fungsi mengirimkan detail tentang setiap invokasi ke CloudWatch Logs. Detail tersebut menyampaikan log yang dihasilkan fungsi Anda selama invokasi. Jika fungsi mengembalikan kesalahan, Lambda memformat kesalahan dan mengembalikannya ke pemanggil.

Topik

- Runtime-included Versi SDK
- Menggunakan keep-alive untuk koneksi TCP
- Pemuatan sertifikat CA
- Eksperimental Node.js fitur
- Tentukan penangan Fungsi Peristiwa Lambda di Node.js
- Menyebarkan Fungsi Acara Node.js Lambda dengan arsip file.zip
- Menyebarkan Fungsi Acara Node.js Lambda dengan gambar kontainer
- Bekerja dengan lapisan untuk Fungsi Node.js Acara Lambda
- Menggunakan objek konteks Lambda untuk mengambil informasi Fungsi Acara Node.js
- Log dan pantau Node.js Fungsi Acara Lambda
- Menginstrumentasi Node.js kode di AWS Lambda

Runtime-included Versi SDK

Semua didukung Lambda Node.js waktu proses termasuk versi minor tertentu dari AWS SDK untuk JavaScript v3, bukan versi terbaru. Versi minor spesifik yang disertakan dalam runtime tergantung pada versi runtime dan versi Anda Wilayah AWS. Untuk menemukan versi spesifik SDK yang disertakan dalam runtime yang Anda gunakan, buat Lambda Event Function dengan kode berikut.

Example index.mjs

```
import packageJson from '@aws-sdk/client-s3/package.json' with { type: 'json' };

export const handler = async () => ({ version: packageJson.version });
```

Ini mengembalikan respons dalam format berikut:

```
{
  "version": "3.632.0"
}
```

Untuk informasi selengkapnya, lihat Menggunakan SDK untuk JavaScript v3 di handler Anda.

Menggunakan keep-alive untuk koneksi TCP

Defaultnya Node.js HTTP/HTTPS agen membuat koneksi TCP baru untuk setiap permintaan baru. Untuk menghindari biaya pembuatan koneksi baru, keep-alive diaktifkan secara default di semua didukung Node.js waktu proses. Keep-alive dapat mengurangi waktu permintaan untuk fungsi Lambda yang membuat beberapa panggilan API menggunakan SDK.

Untuk menonaktifkan keep-alive, lihat Menggunakan kembali koneksi dengan keep-alive in Node.js di AWS SDK untuk JavaScript Panduan Pengembang 3.x. Untuk informasi selengkapnya tentang menggunakan keep-alive, lihat HTTP keep-alive aktif secara default di modular AWS SDK untuk JavaScript pada AWS Blog Alat Pengembang.

Pemuatan sertifikat CA

Untuk Node.js versi runtime hingga Node.js 18, Lambda secara otomatis memuat Amazon-specific Sertifikat CA (otoritas sertifikat) untuk memudahkan Anda membuat fungsi yang berinteraksi dengan yang lain Layanan AWS. Misalnya, Lambda menyertakan sertifikat Amazon RDS yang diperlukan untuk memvalidasi sertifikat identitas server diinstal pada database Amazon RDS Anda. Perilaku ini dapat memiliki dampak kinerja selama start dingin.

Dimulai dengan Node.js 20, Lambda tidak lagi memuat sertifikat CA tambahan secara default. The Node.js 20 runtime berisi file sertifikat dengan semua sertifikat Amazon CA terletak di `/var/runtime/ca-cert.pem`. Untuk mengembalikan perilaku yang sama dari Node.js 18 dan runtime sebelumnya, atur `NODE_EXTRA_CA_CERTS` variabel lingkungan kepada `/var/runtime/ca-cert.pem`.

Untuk kinerja yang optimal, sebaiknya gabungkan hanya sertifikat yang Anda butuhkan dengan paket penerapan Anda dan memuatnya melalui `NODE_EXTRA_CA_CERTS` variabel lingkungan. File sertifikat harus terdiri dari satu atau lebih sertifikat CA root atau perantara tepercaya dalam format PEM. Misalnya, untuk RDS, sertakan sertifikat yang diperlukan di samping kode Anda sebagai `certificates/rds.pem`. Kemudian, muat sertifikat dengan mengatur `NODE_EXTRA_CA_CERTS` kepada `/var/task/certificates/rds.pem`.

Eksperimental Node.js fitur

Hulu Node.js rilis bahasa mengaktifkan beberapa fitur eksperimental secara default. Lambda menonaktifkan fitur-fitur ini untuk memastikan stabilitas runtime dan kinerja yang konsisten. Tabel berikut mencantumkan fitur eksperimental yang dinonaktifkan Lambda.

Fitur eksperimental	Didukung Node.js versi	Node.js bendera diterapkan oleh Lambda	Bendera Lambda untuk mengaktifkan kembali
Dukungan untuk mengimpor modul menggunakan persyaratan dalam modul ES	Node.js 20, Node.js 22, Node.js 24	`--no-experimental-require-module`	`--experimental-require-module`
Dukungan untuk mendeteksi modul ES vs CommonJS secara otomatis	Node.js 22, Node.js 24	`--no-experimental-detect-module`	`--experimental-detect-module`

Untuk mengaktifkan fitur eksperimental yang dinonaktifkan, atur tanda aktifkan kembali di `NODE_OPTIONS` variabel lingkungan. Misalnya, untuk mengaktifkan modul ES memerlukan

dukungan, atur `NODE_OPTIONS` kepada `--experimental-require-module`. Lambda mendeteksi penggantian ini dan menghapus flag nonaktifkan yang sesuai.

> **Important**
>
> Menggunakan fitur eksperimental dapat menyebabkan ketidakstabilan dan masalah kinerja. Fitur-fitur ini mungkin diubah atau dihapus di masa mendatang Node.js versi. Fungsi yang menggunakan fitur eksperimental tidak memenuhi syarat untuk Lambda Service Level Agreement (SLA) atau AWS Dukungan.

Tentukan penangan Fungsi Peristiwa Lambda di Node.js

Lambda Event Function handler adalah metode dalam kode fungsi Anda yang memproses peristiwa. Saat fungsi Anda diaktifkan, Lambda menjalankan metode handler. Fungsi Anda berjalan sampai handler mengembalikan respons, keluar, atau waktu habis.

Halaman ini menjelaskan cara bekerja dengan penangan Fungsi Peristiwa Lambda Node.js, termasuk opsi untuk penyiapan proyek, konvensi penamaan, dan praktik terbaik. Halaman ini juga menyertakan contoh fungsi Node.js Lambda yang mengambil informasi tentang pesanan, menghasilkan tanda terima file teks, dan menempatkan file ini di bucket Amazon Simple Storage Service (Amazon S3). Untuk informasi tentang cara menerapkan fungsi Anda setelah menulisnya, lihat the section called “Deploy arsip file .zip” atauthe section called “Deploy gambar kontainer”.

Topik

- Menyiapkan proyek Node.js handler Anda
- Contoh kode Node.js Fungsi Acara Lambda
- Modul CommonJS dan ES
- Node.js inisialisasi
- Konvensi penamaan handler
- Mendefinisikan dan mengakses objek peristiwa masukan
- Pola handler yang valid untuk fungsi Node.js
- Menggunakan SDK untuk JavaScript v3 di handler Anda
- Mengakses variabel lingkungan
- Menggunakan status global
- Praktik terbaik kode untuk Fungsi Node.js Acara Lambda

Menyiapkan proyek Node.js handler Anda

Ada beberapa cara untuk menginisialisasi proyek Node.js Lambda. Misalnya, Anda dapat membuat Node.js proyek standar menggunakannpm, membuat AWS SAM aplikasi, atau membuat AWS CDK aplikasi.

Untuk membuat proyek menggunakannpm:

```
npm init
```

Perintah ini menginisialisasi proyek Anda dan menghasilkan `package.json` file yang mengelola metadata dan dependensi proyek Anda.

Kode fungsi Anda tinggal di `.mjs` JavaScript file `.js` atau. Dalam contoh berikut, kami menamai file ini `index.mjs` karena menggunakan penangan modul ES. Lambda mendukung modul ES dan penangan CommonJS. Untuk informasi selengkapnya, lihat Modul CommonJS dan ES.

Proyek fungsi Node.js Lambda yang khas mengikuti struktur umum ini:

```
/project-root
  ### index.mjs – Contains main handler
  ### package.json – Project metadata and dependencies
  ### package-lock.json – Dependency lock file
  ### node_modules/ – Installed dependencies
```

Contoh kode Node.js Fungsi Acara Lambda

Contoh kode fungsi Lambda berikut mengambil informasi tentang pesanan, menghasilkan tanda terima file teks, dan menempatkan file ini di bucket Amazon S3.

Example index.mjs Fungsi Acara Lambda

```
import { S3Client, PutObjectCommand } from '@aws-sdk/client-s3';

// Initialize the S3 client outside the handler for reuse
const s3Client = new S3Client();

/**
 * Lambda handler for processing orders and storing receipts in S3.
 * @param {Object} event - Input event containing order details
 * @param {string} event.order_id - The unique identifier for the order
 * @param {number} event.amount - The order amount
 * @param {string} event.item - The item purchased
 * @returns {Promise<string>} Success message
 */
export const handler = async(event) => {
    try {
        // Access environment variables
        const bucketName = process.env.RECEIPT_BUCKET;
        if (!bucketName) {
            throw new Error('RECEIPT_BUCKET environment variable is not set');
        }
```

```
        // Create the receipt content and key destination
        const receiptContent = `OrderID: ${event.order_id}\nAmount: $
${event.amount.toFixed(2)}\nItem: ${event.item}`;
        const key = `receipts/${event.order_id}.txt`;

        // Upload the receipt to S3
        await uploadReceiptToS3(bucketName, key, receiptContent);

        console.log(`Successfully processed order ${event.order_id} and stored receipt
 in S3 bucket ${bucketName}`);
        return 'Success';
    } catch (error) {
        console.error(`Failed to process order: ${error.message}`);
        throw error;
    }
};

/**
 * Helper function to upload receipt to S3
 * @param {string} bucketName - The S3 bucket name
 * @param {string} key - The S3 object key
 * @param {string} receiptContent - The content to upload
 * @returns {Promise<void>}
 */
async function uploadReceiptToS3(bucketName, key, receiptContent) {
    try {
        const command = new PutObjectCommand({
            Bucket: bucketName,
            Key: key,
            Body: receiptContent
        });

        await s3Client.send(command);
    } catch (error) {
        throw new Error(`Failed to upload receipt to S3: ${error.message}`);
    }
}
```

`index.mjs`File ini berisi bagian kode berikut:

- `import`block: Gunakan blok ini untuk menyertakan pustaka yang dibutuhkan fungsi Lambda Anda, AWS seperti klien SDK.

- `const s3Client`deklarasi: Ini menginisialisasi klien Amazon S3 di luar fungsi handler. Hal ini menyebabkan Lambda menjalankan kode ini selama fase inisialisasi, dan klien dipertahankan untuk digunakan kembali di beberapa pemanggilan.
- Blok komentar JSDoc: Tentukan jenis input dan output untuk handler Anda menggunakan anotasi JSDoc.
- `export const handler`: Ini adalah fungsi handler utama yang dipanggil Lambda. Saat menerapkan fungsi Anda, tentukan properti `index.handler` Handler. Nilai `Handler` properti adalah nama file dan nama metode handler yang diekspor, dipisahkan oleh sebuah titik.
- `uploadReceiptToS3`fungsi: Ini adalah fungsi pembantu yang direferensikan oleh fungsi handler utama.

Agar fungsi ini berfungsi dengan baik, peran pelaksanaannya harus memungkinkan `s3:PutObject` tindakan. Juga, pastikan bahwa Anda mendefinisikan variabel `RECEIPT_BUCKET` lingkungan. Setelah pemanggilan berhasil, bucket Amazon S3 harus berisi file tanda terima.

Modul CommonJS dan ES

Node.js mendukung dua sistem modul: modul CommonJS dan ECMAScript (modul ES). Lambda merekomendasikan penggunaan modul ES karena mendukung await tingkat atas, yang memungkinkan tugas asinkron diselesaikan selama inisialisasi lingkungan eksekusi.

Node.js memperlakukan file dengan ekstensi nama `.cjs` file sebagai modul CommonJS sementara `.mjs` ekstensi menunjukkan modul ES. Secara default, Node.js memperlakukan file dengan ekstensi nama `.js` file sebagai modul CommonJS. Anda dapat mengonfigurasi Node.js untuk memperlakukan `.js` file sebagai modul ES dengan menentukan `type` as `module` pada `package.json` file fungsi. Anda dapat mengonfigurasi Node.js di Lambda untuk mendeteksi secara otomatis apakah `.js` file harus diperlakukan sebagai CommonJS atau sebagai modul ES dengan menambahkan `–experimental-detect-module` flag ke variabel lingkungan. `NODE_OPTIONS` Untuk informasi selengkapnya, lihat Node.js Fitur eksperimental.

Contoh berikut menunjukkan penangan fungsi yang ditulis menggunakan modul ES dan modul CommonJS. Contoh yang tersisa di halaman ini semuanya menggunakan modul ES.

ES module example

Example- Penangan modul ES

```
const url = "https://aws.amazon.com/";
```

```
export const handler = async(event) => {
    try {
        const res = await fetch(url);
        console.info("status", res.status);
        return res.status;
    }
    catch (e) {
        console.error(e);
        return 500;
    }
};
```

CommonJS module example

Example- Penangan modul CommonJS

```
const https = require("https");
let url = "https://aws.amazon.com/";

exports.handler = async function (event) {
  let statusCode;
  await new Promise(function (resolve, reject) {
    https.get(url, (res) => {
        statusCode = res.statusCode;
        resolve(statusCode);
      }).on("error", (e) => {
        reject(Error(e));
      });
  });
  console.log(statusCode);
  return statusCode;
};
```

Node.js inisialisasi

Node.js menggunakan I/O model non-pemblokiran yang mendukung operasi asinkron yang efisien menggunakan loop peristiwa. Misalnya, jika Node.js melakukan panggilan jaringan, fungsi terus memproses operasi lain tanpa memblokir respons jaringan. Ketika respon jaringan diterima, itu ditempatkan ke antrian callback. Tugas dari antrian diproses ketika tugas saat ini selesai.

Lambda merekomendasikan penggunaan await tingkat atas sehingga tugas asinkron yang dimulai selama inisialisasi lingkungan eksekusi diselesaikan selama inisialisasi. Tugas asinkron yang

tidak selesai selama inisialisasi biasanya berjalan selama pemanggilan fungsi pertama. Ini dapat menyebabkan perilaku atau kesalahan yang tidak terduga.

Misalnya, inisialisasi fungsi Anda mungkin membuat panggilan jaringan untuk mengambil parameter dari AWS Parameter Store. Jika tugas ini tidak selesai selama inisialisasi, nilainya mungkin null selama pemanggilan. Mungkin juga ada penundaan antara inisialisasi dan pemanggilan yang dapat memicu kesalahan dalam operasi yang sensitif terhadap waktu. Secara khusus, panggilan AWS layanan dapat mengandalkan tanda tangan permintaan yang sensitif terhadap waktu, yang mengakibatkan kegagalan panggilan layanan jika panggilan tidak selesai selama fase inisialisasi.

Menyelesaikan tugas selama inisialisasi biasanya meningkatkan kinerja cold-start, dan pertama-tama memanggil kinerja saat menggunakan Provisioned Concurrency. Untuk informasi lebih lanjut, lihat posting blog kami Menggunakan modul Node.js ES dan menunggu tingkat atas. AWS Lambda

Konvensi penamaan handler

Saat Anda mengonfigurasi fungsi, nilai pengaturan Handler adalah nama file dan nama metode handler yang diekspor, dipisahkan oleh titik. Default untuk fungsi yang dibuat di konsol dan untuk contoh dalam panduan ini adalah`index.handler`. Ini menunjukkan `handler` metode yang diekspor dari `index.mjs` file `index.js` atau.

Jika Anda membuat fungsi di konsol menggunakan nama file atau nama pengendali fungsi yang berbeda, Anda harus mengedit nama handler default.

Untuk mengubah nama fungsi handler (konsol)

1. Buka halaman Fungsi di konsol Lambda dan pilih fungsi Anda.
2. Pilih tab Kode.
3. Gulir ke bawah ke panel pengaturan Runtime dan pilih Edit.
4. Di Handler, masukkan nama baru untuk handler fungsi Anda.
5. Pilih Simpan.

Mendefinisikan dan mengakses objek peristiwa masukan

JSON adalah format input yang paling umum dan standar untuk fungsi Lambda. Dalam contoh ini, fungsi mengharapkan input yang mirip dengan yang berikut:

```
{
```

```
    "order_id": "12345",
    "amount": 199.99,
    "item": "Wireless Headphones"
}
```

Saat bekerja dengan fungsi Lambda di Node.js, Anda dapat menentukan bentuk yang diharapkan dari peristiwa input menggunakan anotasi JSDoc. Dalam contoh ini, kita mendefinisikan struktur input dalam komentar JSDoc handler:

```
/**
 * Lambda handler for processing orders and storing receipts in S3.
 * @param {Object} event - Input event containing order details
 * @param {string} event.order_id - The unique identifier for the order
 * @param {number} event.amount - The order amount
 * @param {string} event.item - The item purchased
 * @returns {Promise<string>} Success message
 */
```

Setelah Anda menentukan jenis ini dalam komentar JSDoc Anda, Anda dapat mengakses bidang objek acara langsung dalam kode Anda. Misalnya, `event.order_id` mengambil nilai dari `order_id` dari input asli.

Pola handler yang valid untuk fungsi Node.js

Sebaiknya gunakan async/awaituntuk mendeklarasikan fungsi handler alih-alih menggunakan callback. Async/await adalah cara ringkas dan mudah dibaca untuk menulis kode asinkron, tanpa perlu panggilan balik bersarang atau janji rantai. Dengan async/await, Anda dapat menulis kode yang berbunyi seperti kode sinkron, sambil tetap asinkron dan tidak memblokir.

penangan fungsi async (disarankan)

`async`Kata kunci menandai fungsi sebagai asinkron, dan `await` kata kunci menjeda eksekusi fungsi sampai a diselesaikan. `Promise` Handler menerima argumen berikut:

- `event`: Berisi data input yang diteruskan ke fungsi Anda.
- `context`: Berisi informasi tentang pemanggilan, fungsi, dan lingkungan eksekusi. Untuk informasi selengkapnya, lihat Menggunakan objek konteks Lambda untuk mengambil informasi Fungsi Acara Node.js.

Berikut adalah tanda tangan yang valid untuk pola tersebut async/await :

```
export const handler = async (event) => { };
```

```
export const handler = async (event, context) => { };
```

Penangan fungsi sinkron

Jika fungsi Anda tidak melakukan tugas asinkron, Anda dapat menggunakan penangan fungsi sinkron, menggunakan salah satu tanda tangan fungsi berikut:

```
export const handler = (event) => { };
```

```
export const handler = (event, context) => { };
```

Respon penangan fungsi streaming

Lambda mendukung streaming respons dengan. Node.js Response streaming function handler menggunakan `awslambda.streamifyResponse()` dekorator dan mengambil 3 parameter:`event`,`responseStream`, dan. `context` Tanda tangan fungsi adalah:

```
export const handler = awslambda.streamifyResponse(async (event, responseStream,
 context) => { });
```

Untuk informasi selengkapnya, lihat Streaming respons untuk Fungsi Acara Lambda.

Callback-based penangan fungsi

> **Note**
>
> Callback-based penangan fungsi hanya didukung hingga Node.js 22. Mulai dari Node.js 24, tugas asinkron harus diimplementasikan menggunakan penangan fungsi asinkron.

Callback-based penangan fungsi harus menggunakan argumen event, context, dan callback. Contoh:

```
export const handler = (event, context, callback) => { };
```

Fungsi callback mengharapkan `Error` dan respons, yang harus. JSON-serializable Fungsi terus mengeksekusi sampai loop peristiwa kosong atau fungsi habis waktu. Respons tidak dikirimkan

ke invoker hingga semua tugas loop peristiwa selesai. Jika waktu fungsi habis, kesalahan akan dikembalikan. Anda dapat mengonfigurasi runtime untuk mengirim respons segera dengan menyetel WaitsForEmptyEventLoopcontext.callback ke false.

Example— Permintaan HTTP dengan callback

Fungsi contoh berikut memeriksa URL dan mengembalikan kode status kepada invoker.

```
import https from "https";
let url = "https://aws.amazon.com/";

export const handler = (event, context, callback) => {
  https.get(url, (res) => {
    callback(null, res.statusCode);
  }).on("error", (e) => {
    callback(Error(e));
  });
};
```

Menggunakan SDK untuk JavaScript v3 di handler Anda

Seringkali, Anda akan menggunakan fungsi Lambda untuk berinteraksi dengan atau membuat pembaruan ke sumber daya lain AWS . Cara termudah untuk berinteraksi dengan sumber daya ini adalah dengan menggunakan file AWS SDK untuk JavaScript. Semua Node.js runtime Lambda yang didukung menyertakan SDK untuk versi 3. JavaScript

Namun, kami sangat menyarankan agar Anda menyertakan klien AWS SDK yang Anda butuhkan dalam paket penerapan Anda. Ini memaksimalkan kompatibilitas mundur selama pembaruan runtime Lambda di masa mendatang. Hanya mengandalkan SDK yang disediakan runtime jika Anda tidak dapat menyertakan paket tambahan (misalnya, saat menggunakan editor kode konsol Lambda atau kode sebaris dalam templat). AWS CloudFormation

Untuk menambahkan dependensi SDK ke fungsi Anda, gunakan `npm install` perintah untuk klien SDK tertentu yang Anda butuhkan. Dalam kode contoh, kami menggunakan klien Amazon S3. Tambahkan dependensi ini dengan menjalankan perintah berikut di direktori yang berisi `package.json` file Anda:

```
npm install @aws-sdk/client-s3
```

Dalam kode fungsi, impor klien dan perintah yang Anda butuhkan, seperti yang ditunjukkan oleh fungsi contoh:

```
import { S3Client, PutObjectCommand } from '@aws-sdk/client-s3';
```

Kemudian, inisialisasi klien Amazon S3:

```
const s3Client = new S3Client();
```

Dalam contoh ini, kami menginisialisasi klien Amazon S3 kami di luar fungsi pengendali utama untuk menghindari keharusan menginisialisasi setiap kali kami menjalankan fungsi kami. Setelah menginisialisasi klien SDK, Anda dapat menggunakannya untuk melakukan panggilan API untuk layanan tersebut AWS . Kode contoh memanggil tindakan PutObjectAPI Amazon S3 sebagai berikut:

```
const command = new PutObjectCommand({
    Bucket: bucketName,
    Key: key,
    Body: receiptContent
});
```

Mengakses variabel lingkungan

Dalam kode handler Anda, Anda dapat mereferensikan variabel lingkungan apa pun dengan menggunakan`process.env`. Dalam contoh ini, kami mereferensikan variabel `RECEIPT_BUCKET` lingkungan yang ditentukan menggunakan baris kode berikut:

```
// Access environment variables
const bucketName = process.env.RECEIPT_BUCKET;
if (!bucketName) {
    throw new Error('RECEIPT_BUCKET environment variable is not set');
}
```

Menggunakan status global

Lambda menjalankan kode statis Anda selama fase inisialisasi sebelum menjalankan fungsi Anda untuk pertama kalinya. Sumber daya yang dibuat selama inisialisasi tetap berada di memori di antara pemanggilan, sehingga Anda dapat menghindari keharusan membuatnya setiap kali Anda menjalankan fungsi Anda.

Dalam kode contoh, kode inisialisasi klien S3 berada di luar handler. Runtime menginisialisasi klien sebelum fungsi menangani peristiwa pertamanya, dan klien tetap tersedia untuk digunakan kembali di semua pemanggilan.

Praktik terbaik kode untuk Fungsi Node.js Acara Lambda

Ikuti panduan ini saat membangun fungsi Lambda:

- Pisahkan handler Lambda dari logika inti Anda. Ini memungkinkan Anda untuk membuat fungsi yang lebih dapat teruji.
- Kontrol dependensi dalam paket penerapan fungsi Anda. Lingkungan AWS Lambda eksekusi berisi sejumlah pustaka. Untuk runtime Node.js dan Python, ini termasuk SDK. AWS Untuk mengaktifkan serangkaian fitur dan pembaruan keamanan terbaru, Lambda akan memperbarui pustaka ini secara berkala. Pembaruan ini dapat memberikan perubahan kecil pada perilaku fungsi Lambda Anda. Untuk memiliki kendali penuh atas dependensi yang digunakan fungsi Anda, kemas semua dependensi Anda dengan paket deployment Anda.
- Minimalkan kompleksitas dependensi Anda. Utamakan memilih kerangka kerja lebih sederhana yang cepat dimuat dalam memulai lingkungan eksekusi.
- Minimalkan ukuran paket penerapan Anda sesuai kebutuhan runtime-nya. Ini akan mengurangi jumlah waktu yang dibutuhkan untuk mengunduh dan membongkar paket deployment Anda sebelum invokasi.

Manfaatkan penggunaan kembali lingkungan eksekusi untuk meningkatkan kinerja fungsi Anda. Inisialisasi klien SDK dan koneksi basis data di luar fungsi handler, dan lakukan caching aset statis secara lokal di direktori `/tmp`. Invokasi selanjutnya yang diproses oleh instans yang sama dari fungsi Anda dapat menggunakan kembali sumber daya ini. Ini menghemat biaya dengan mengurangi waktu pengoperasian fungsi.

Untuk menghindari potensi kebocoran data di seluruh invokasi, jangan menggunakan lingkungan eksekusi untuk menyimpan data pengguna, peristiwa, atau informasi lainnya implikasi keamanan. Jika fungsi Anda bergantung pada status yang dapat disenyapkan yang tidak dapat disimpan dalam memori di dalam handler, pertimbangkan untuk membuat fungsi terpisah atau versi terpisah dari fungsi untuk setiap pengguna.

Gunakan arahan keep-alive untuk mempertahankan koneksi yang persisten. Lambda membersihkan koneksi idle dari waktu ke waktu. Mencoba menggunakan ulang koneksi idle saat mengidentifikasi suatu fungsi akan menyebabkan kesalahan koneksi. Untuk mempertahankan koneksi yang persisten, gunakan arahan tetap aktif yang berkaitan dengan runtime Anda. Sebagai contoh, lihat Menggunakan Kembali Koneksi dengan Keep-Alive in Node.js.

Gunakan variabel lingkungan untuk meneruskan parameter operasional ke fungsi Anda. Misalnya, jika Anda ingin menulis ke bucket Amazon S3 alih-alih melakukan hard-coding nama bucket yang Anda tulis, konfigurasikan nama bucket sebagai variabel lingkungan.

Hindari menggunakan pemanggilan rekursif dalam fungsi Lambda Anda, di mana fungsi memanggil dirinya sendiri atau memulai proses yang dapat memanggil fungsi lagi. Hal ini dapat menyebabkan volume invokasi fungsi yang tidak diinginkan dan peningkatan biaya. Jika Anda melihat volume pemanggilan yang tidak diinginkan, setel konkurensi fungsi cadangan untuk `0` segera membatasi semua pemanggilan ke fungsi, saat Anda memperbarui kode.

Jangan gunakan API non-publik yang tidak terdokumentasi dalam kode fungsi Lambda Anda. Untuk runtime AWS Lambda terkelola, Lambda secara berkala menerapkan pembaruan keamanan dan fungsional ke API internal Lambda. Pembaruan API internal ini mungkin tidak kompatibel ke belakang, yang menyebabkan konsekuensi yang tidak diinginkan seperti kegagalan pemanggilan jika fungsi Anda memiliki ketergantungan pada API non-publik ini. Lihat referensi API untuk daftar API yang tersedia untuk umum.

Tulis kode idempoten. Menulis kode idempoten untuk fungsi Anda memastikan bahwa peristiwa duplikat ditangani dengan cara yang sama. Kode Anda harus memvalidasi peristiwa dengan benar dan menangani peristiwa duplikat dengan anggun. Untuk informasi selengkapnya, lihat Bagaimana cara membuat fungsi Lambda saya idempoten? .

Menyebarkan Fungsi Acara Node.js Lambda dengan arsip file.zip

Kode AWS Lambda fungsi Anda terdiri dari file.js atau .mjs yang berisi kode handler fungsi Anda, bersama dengan paket dan modul tambahan yang bergantung pada kode Anda. Untuk menyebarkan kode fungsi ini ke Lambda, Anda menggunakan paket penerapan. Paket ini dapat berupa arsip file.zip atau gambar kontainer. Untuk informasi selengkapnya tentang penggunaan gambar kontainer dengan Node.js, lihat Menerapkan Fungsi Peristiwa Node.js Lambda dengan gambar kontainer.

Untuk membuat paket penyebaran Anda sebagai arsip file.zip, Anda dapat menggunakan utilitas arsip file.zip bawaan alat baris perintah Anda, atau utilitas file.zip lainnya seperti 7zip. Contoh yang ditampilkan di bagian berikut mengasumsikan Anda menggunakan `zip` alat baris perintah di lingkungan Linux atau macOS. Untuk menggunakan perintah yang sama di Windows, Anda dapat menginstal Subsistem Windows untuk Linux untuk mendapatkan Windows-integrated versi Ubuntu dan Bash.

Perhatikan bahwa Lambda menggunakan izin file POSIX, jadi Anda mungkin perlu mengatur izin untuk folder paket penyebaran sebelum membuat arsip file.zip.

Topik

- Dependensi runtime di Node.js
- Membuat paket penerapan.zip tanpa dependensi
- Membuat paket penerapan.zip dengan dependensi
- Membuat Node.js layer untuk dependensi Anda
- Jalur pencarian ketergantungan dan pustaka yang disertakan runtime
- Membuat dan memperbarui Fungsi Acara Node.js Lambda menggunakan file.zip

Dependensi runtime di Node.js

Untuk Lambda Event Functions yang menggunakan Node.js runtime, dependensi dapat berupa modul apa pun. Node.js Node.js Runtime mencakup sejumlah pustaka umum, serta versi. AWS SDK untuk JavaScript Semua Node.js runtime yang didukung menyertakan SDK versi 3. Untuk menggunakan SDK versi 2, Anda harus menambahkan SDK ke paket penyebaran file.zip Anda. Untuk menemukan versi spesifik SDK yang disertakan dalam runtime yang Anda gunakan, lihat. the section called “Runtime-included Versi SDK”

Lambda memperbarui pustaka SDK secara berkala di Node.js runtime untuk menyertakan fitur terbaru dan peningkatan keamanan. Lambda juga menerapkan patch keamanan dan pembaruan

ke pustaka lain yang termasuk dalam runtime. Untuk memiliki kontrol penuh atas dependensi dalam paket Anda, Anda dapat menambahkan versi pilihan Anda dari setiap dependensi yang disertakan runtime ke paket penerapan Anda. Misalnya, jika Anda ingin menggunakan versi SDK tertentu JavaScript, Anda dapat memasukkannya ke dalam file.zip sebagai dependensi. Untuk informasi selengkapnya tentang menambahkan dependensi yang disertakan runtime ke file.zip Anda, lihat. Jalur pencarian ketergantungan dan pustaka yang disertakan runtime

Di bawah model tanggung jawab AWS bersama, Anda bertanggung jawab atas pengelolaan dependensi apa pun dalam paket penerapan fungsi Anda. Ini termasuk menerapkan pembaruan dan patch keamanan. Untuk memperbarui dependensi dalam paket penerapan fungsi Anda, pertama buat file.zip baru dan kemudian unggah ke Lambda. Lihat Membuat paket penerapan.zip dengan dependensi dan Membuat dan memperbarui Fungsi Acara Node.js Lambda menggunakan file.zip untuk informasi lebih lanjut.

Membuat paket penerapan.zip tanpa dependensi

Jika kode fungsi Anda tidak memiliki dependensi kecuali pustaka yang disertakan dalam runtime Lambda, file.zip Anda hanya berisi `index.mjs` file `index.js` atau dengan kode handler fungsi Anda. Gunakan utilitas zip pilihan Anda untuk membuat file.zip dengan `index.mjs` file `index.js` atau Anda di root. Jika file yang berisi kode handler Anda tidak berada di root file.zip Anda, Lambda tidak akan dapat menjalankan kode Anda.

Untuk mempelajari cara menerapkan file.zip Anda untuk membuat fungsi Lambda baru atau memperbarui yang sudah ada, lihat. Membuat dan memperbarui Fungsi Acara Node.js Lambda menggunakan file.zip

Membuat paket penerapan.zip dengan dependensi

Jika kode fungsi Anda bergantung pada paket atau modul yang tidak disertakan dalam Node.js runtime Lambda, Anda dapat menambahkan dependensi ini ke file.zip dengan kode fungsi Anda atau menggunakan lapisan Lambda. Petunjuk di bagian ini menunjukkan cara memasukkan dependensi Anda ke dalam paket penerapan.zip Anda. Untuk petunjuk tentang cara memasukkan dependensi Anda dalam lapisan, lihat. the section called “Membuat Node.js layer untuk dependensi Anda”

Contoh perintah CLI berikut membuat file.zip bernama `my_deployment_package.zip` berisi `index.mjs` file `index.js` or dengan kode handler fungsi Anda dan dependensinya. Dalam contoh, Anda menginstal dependensi menggunakan manajer paket npm.

Untuk membuat paket deployment

1. Arahkan ke direktori proyek yang berisi file kode `index.mjs` sumber `index.js` atau Anda. Dalam contoh ini, direktori diberi namamy_`function`.

   ```
   cd my_function
   ```

2. Instal pustaka yang diperlukan fungsi Anda di `node_modules` direktori menggunakan `npm install` perintah. Dalam contoh ini Anda menginstal file AWS X-Ray SDK for Node.js.

   ```
   npm install aws-xray-sdk
   ```

   Ini menciptakan struktur folder yang mirip dengan berikut ini:

   ```
   ~/my_function
   ### index.mjs
   ### node_modules
       ### async
       ### async-listener
       ### atomic-batcher
       ### aws-sdk
       ### aws-xray-sdk
       ### aws-xray-sdk-core
   ```

   Anda juga dapat menambahkan modul khusus yang Anda buat sendiri ke paket penyebaran Anda. Buat direktori di bawah `node_modules` dengan nama modul Anda dan simpan paket tertulis kustom Anda di sana.

3. Buat file.zip yang berisi isi folder proyek Anda di root. Gunakan opsi `r` (rekursif) untuk memastikan zip mengompresi subfolder.

   ```
   zip -r my_deployment_package.zip .
   ```

Membuat Node.js layer untuk dependensi Anda

Instruksi di bagian ini menunjukkan kepada Anda bagaimana memasukkan dependensi Anda dalam lapisan. Untuk petunjuk tentang cara menyertakan dependensi Anda dalam paket penerapan Anda, lihat. the section called “Membuat paket penerapan.zip dengan dependensi” Untuk informasi selengkapnya tentang membuat dan mengelola Node.js lapisan, lihatthe section called “Lapisan”.

Saat Anda menambahkan lapisan ke fungsi, Lambda memuat konten lapisan ke dalam `/opt` direktori lingkungan eksekusi itu. Untuk setiap runtime Lambda, `PATH` variabel sudah menyertakan jalur folder tertentu dalam direktori. `/opt` Untuk memastikan bahwa Lambda mengambil konten layer Anda, file layer.zip Anda harus memiliki dependensinya di salah satu jalur folder berikut:

- `nodejs/node_modules`
- `nodejs/node18/node_modules (NODE_PATH)`
- `nodejs/node20/node_modules (NODE_PATH)`
- `nodejs/node22/node_modules (NODE_PATH)`

Misalnya, struktur file layer.zip Anda mungkin terlihat seperti berikut:

```
xray-sdk.zip
# nodejs/node_modules/aws-xray-sdk
```

Selain itu, Lambda secara otomatis mendeteksi pustaka apa pun di `/opt/lib` direktori, dan binari apa pun di direktori. `/opt/bin` Untuk memastikan bahwa Lambda menemukan konten layer Anda dengan benar, Anda juga dapat membuat layer dengan struktur berikut:

```
custom-layer.zip
# lib
    | lib_1
    | lib_2
# bin
    | bin_1
    | bin_2
```

Setelah Anda mengemas layer Anda, lihat the section called “Membuat dan menghapus lapisan” dan the section called “Menambahkan lapisan” untuk menyelesaikan setup layer Anda.

Jalur pencarian ketergantungan dan pustaka yang disertakan runtime

Node.js Runtime mencakup sejumlah pustaka umum, serta versi. AWS SDK untuk JavaScript Jika Anda ingin menggunakan versi yang berbeda dari pustaka yang disertakan runtime, Anda dapat melakukannya dengan menggabungkannya dengan fungsi Anda atau dengan menambahkannya sebagai dependensi dalam paket penerapan Anda. Misalnya, Anda dapat menggunakan versi SDK yang berbeda dengan menambahkannya ke paket deployment .zip Anda. Anda juga dapat memasukkannya ke dalam lapisan Lambda untuk fungsi Anda.

Saat Anda menggunakan `require` pernyataan `import` atau dalam kode Anda, Node.js runtime akan mencari direktori di `NODE_PATH` jalur hingga menemukan modul. Secara default, lokasi pertama pencarian runtime adalah direktori tempat paket deployment .zip Anda didekompresi dan dipasang (). `/var/task` Jika Anda menyertakan versi pustaka yang disertakan runtime dalam paket penerapan, versi ini lebih diutamakan daripada versi yang disertakan dalam runtime. Dependensi dalam paket penerapan Anda juga lebih diutamakan daripada dependensi dalam lapisan.

Saat Anda menambahkan dependensi ke lapisan, Lambda mengekstrak ini `/opt/nodejs/nodexx/node_modules` ke `nodexx` mana mewakili versi runtime yang Anda gunakan. Di jalur pencarian, direktori ini lebih diutamakan daripada direktori yang berisi pustaka yang disertakan runtime (). `/var/lang/lib/node_modules` Oleh karena itu, pustaka di lapisan fungsi lebih diutamakan daripada versi yang disertakan dalam runtime.

Anda dapat melihat jalur pencarian lengkap untuk fungsi Lambda Anda dengan menambahkan baris kode berikut.

```
console.log(process.env.NODE_PATH)
```

Anda juga dapat menambahkan dependensi di folder terpisah di dalam paket.zip Anda. Misalnya, Anda dapat menambahkan modul kustom ke folder dalam paket.zip Anda yang dipanggil`common`. Ketika paket.zip Anda didekompresi dan dipasang, folder ini ditempatkan di dalam direktori. `/var/task` Untuk menggunakan dependensi dari folder dalam paket penyebaran .zip Anda dalam kode Anda, gunakan `const { } = require()` pernyataan `import { } from` atau, tergantung pada apakah Anda menggunakan resolusi modul CJS atau ESM. Contoh:

```
import { myModule } from './common'
```

Jika Anda menggabungkan kode Anda dengan`esbuild`,`rollup`, atau serupa, dependensi yang digunakan oleh fungsi Anda dibundel bersama dalam satu atau beberapa file. Kami merekomendasikan menggunakan metode ini untuk menjual dependensi bila memungkinkan. Dibandingkan dengan menambahkan dependensi ke paket penerapan Anda, bundling kode Anda menghasilkan peningkatan kinerja karena pengurangan operasi. I/O

Membuat dan memperbarui Fungsi Acara Node.js Lambda menggunakan file.zip

Setelah Anda membuat paket.zip deployment, Anda dapat menggunakannya untuk membuat fungsi Lambda baru atau memperbarui yang sudah ada. Anda dapat menerapkan paket.zip Anda menggunakan konsol Lambda, API, AWS Command Line Interface dan Lambda. Anda juga dapat

membuat dan memperbarui fungsi Lambda menggunakan AWS Serverless Application Model (AWS SAM) dan. CloudFormation

Ukuran maksimum untuk paket.zip deployment untuk Lambda adalah 250 MB (unzip). Perhatikan bahwa batas ini berlaku untuk ukuran gabungan semua file yang Anda unggah, termasuk lapisan Lambda apa pun.

Runtime Lambda membutuhkan izin untuk membaca file dalam paket deployment Anda. Dalam notasi oktal izin Linux, Lambda membutuhkan 644 izin untuk file yang tidak dapat dieksekusi (rw-r--r--) dan 755 izin (rwxr-xr-x) untuk direktori dan file yang dapat dieksekusi.

Di Linux dan macOS, gunakan `chmod` perintah untuk mengubah izin file pada file dan direktori dalam paket penyebaran Anda. Misalnya, untuk memberikan file yang tidak dapat dieksekusi izin yang benar, jalankan perintah berikut.

```
chmod 644 <filepath>
```

Untuk mengubah izin file di Windows, lihat Mengatur, Melihat, Mengubah, atau Menghapus Izin pada Objek dalam dokumentasi Microsoft Windows.

> **Note**
>
> Jika Anda tidak memberikan Lambda izin yang diperlukan untuk mengakses direktori dalam paket penerapan Anda, Lambda menetapkan izin untuk direktori tersebut ke 755 (rwxr-xr-x).

Membuat dan memperbarui fungsi dengan file.zip menggunakan konsol

Untuk membuat fungsi baru, Anda harus terlebih dahulu membuat fungsi di konsol, lalu mengunggah arsip.zip Anda. Untuk memperbarui fungsi yang ada, buka halaman untuk fungsi Anda, lalu ikuti prosedur yang sama untuk menambahkan file.zip Anda yang diperbarui.

Jika file.zip Anda kurang dari 50MB, Anda dapat membuat atau memperbarui fungsi dengan mengunggah file langsung dari mesin lokal Anda. Untuk file.zip yang lebih besar dari 50MB, Anda harus mengunggah paket Anda ke bucket Amazon S3 terlebih dahulu. Untuk petunjuk tentang cara mengunggah file ke bucket Amazon S3 menggunakan Konsol Manajemen AWS, lihat Memulai Amazon S3. Untuk mengunggah file menggunakan AWS CLI, lihat Memindahkan objek di Panduan AWS CLI Pengguna.

> **Note**
>
> Anda tidak dapat mengubah jenis paket penerapan (.zip atau image kontainer) untuk fungsi yang ada. Misalnya, Anda tidak dapat mengonversi fungsi gambar kontainer untuk menggunakan arsip file.zip. Anda harus membuat fungsi baru.

Untuk membuat fungsi baru (konsol)

1. Buka halaman Fungsi Acara konsol Lambda dan pilih Buat Fungsi Acara.
2. Pilih Tulis dari awal.
3. Di bagian Informasi dasar, lakukan hal berikut:
   a. Untuk nama Fungsi, masukkan nama untuk fungsi Anda.
   b. Untuk Runtime, pilih runtime yang ingin Anda gunakan.
   c. (Opsional) Untuk Arsitektur, pilih arsitektur set instruksi untuk fungsi Anda. Arsitektur defaultnya adalah x86_64. Pastikan bahwa paket deployment .zip untuk fungsi Anda kompatibel dengan arsitektur set instruksi yang Anda pilih.
4. (Opsional) Di bagian Izin, luaskan Ubah peran eksekusi default. Anda dapat membuat peran Eksekusi baru atau menggunakan yang sudah ada.
5. Pilih Create Event Function. Lambda menciptakan fungsi dasar 'Hello world' menggunakan runtime yang Anda pilih.

Untuk mengunggah arsip.zip dari mesin lokal Anda (konsol)

1. Di halaman Fungsi Acara konsol Lambda, pilih fungsi yang ingin Anda unggah file.zip.
2. Pilih tab Kode.
3. Di panel Sumber kode, pilih Unggah dari.
4. Pilih file.zip.
5. Untuk mengunggah file.zip, lakukan hal berikut:
   a. Pilih Unggah, lalu pilih file.zip Anda di pemilih file.
   b. Pilih Buka.
   c. Pilih Simpan.

Untuk mengunggah arsip.zip dari bucket Amazon S3 (konsol)

1. Di halaman Fungsi Acara konsol Lambda, pilih fungsi yang ingin Anda unggah file.zip baru.
2. Pilih tab Kode.
3. Di panel Sumber kode, pilih Unggah dari.
4. Pilih lokasi Amazon S3.
5. Rekatkan URL tautan Amazon S3 dari file.zip Anda dan pilih Simpan.

Memperbarui fungsi file.zip menggunakan editor kode konsol

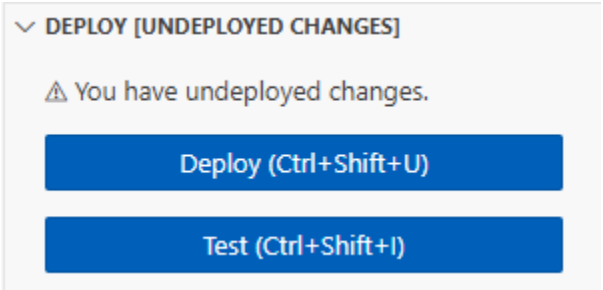
Untuk beberapa fungsi dengan paket penyebaran.zip, Anda dapat menggunakan editor kode bawaan konsol Lambda untuk memperbarui kode fungsi Anda secara langsung. Untuk menggunakan fitur ini, fungsi Anda harus memenuhi kriteria berikut:

- Fungsi Anda harus menggunakan salah satu runtime bahasa yang ditafsirkan (Python,, Node.js atau Ruby)
- Paket penerapan fungsi Anda harus lebih kecil dari 50 MB (unzip).

Kode fungsi untuk fungsi dengan paket penerapan gambar kontainer tidak dapat diedit langsung di konsol.

Untuk memperbarui kode fungsi menggunakan editor kode konsol

1. Buka halaman Fungsi Acara konsol Lambda dan pilih fungsi Anda.
2. Pilih tab Kode.
3. Di panel Sumber kode, pilih file kode sumber Anda dan edit di editor kode terintegrasi.
4. Di bagian DEPLOY, pilih Deploy untuk memperbarui kode fungsi Anda:



Membuat dan memperbarui fungsi dengan file.zip menggunakan AWS CLI

Anda dapat menggunakan AWS CLIuntuk membuat fungsi baru atau memperbarui yang sudah ada menggunakan file.zip. Gunakan perintah create-function dan update-function-code untuk menyebarkan paket.zip Anda. Jika file.zip Anda lebih kecil dari 50MB, Anda dapat mengunggah paket.zip dari lokasi file di mesin build lokal Anda. Untuk file yang lebih besar, Anda harus mengunggah paket.zip Anda dari bucket Amazon S3. Untuk petunjuk tentang cara mengunggah file ke bucket Amazon S3 menggunakan AWS CLI, lihat Memindahkan objek di AWS CLI Panduan Pengguna.

> **Note**
>
> Jika Anda mengunggah file.zip dari bucket Amazon S3 menggunakan AWS CLI, bucket harus berada di lokasi yang Wilayah AWS sama dengan fungsi Anda.

Untuk membuat fungsi baru menggunakan file.zip dengan AWS CLI, Anda harus menentukan yang berikut:

- Nama fungsi Anda (`--function-name`)
- Runtime () `--runtime` fungsi Anda
- Nama Sumber Daya Amazon (ARN) dari peran eksekusi fungsi Anda () `--role`
- Nama metode handler dalam kode fungsi Anda () `--handler`

Anda juga harus menentukan lokasi file.zip Anda. Jika file.zip Anda terletak di folder di mesin build lokal Anda, gunakan `--zip-file` opsi untuk menentukan jalur file, seperti yang ditunjukkan pada perintah contoh berikut.

```
aws lambda create-function --function-name myFunction \
--runtime nodejs24.x --handler index.handler \
--role arn:aws:iam::111122223333:role/service-role/my-lambda-role \
--zip-file fileb://myFunction.zip
```

Untuk menentukan lokasi file.zip di bucket Amazon S3, gunakan opsi seperti `--code` yang ditunjukkan pada perintah contoh berikut. Anda hanya perlu menggunakan `S3ObjectVersion` parameter untuk objek berversi.

```
aws lambda create-function --function-name myFunction \
--runtime nodejs24.x --handler index.handler \
```

```
--role arn:aws:iam::111122223333:role/service-role/my-lambda-role \
--code S3Bucket=amzn-s3-demo-
bucket,S3Key=myFileName.zip,S3ObjectVersion=myObjectVersion
```

Untuk memperbarui fungsi yang ada menggunakan CLI, Anda menentukan nama fungsi Anda menggunakan parameter. `--function-name` Anda juga harus menentukan lokasi file.zip yang ingin Anda gunakan untuk memperbarui kode fungsi Anda. Jika file.zip Anda terletak di folder di mesin build lokal Anda, gunakan `--zip-file` opsi untuk menentukan jalur file, seperti yang ditunjukkan pada perintah contoh berikut.

```
aws lambda update-function-code --function-name myFunction \
--zip-file fileb://myFunction.zip
```

Untuk menentukan lokasi file.zip di bucket Amazon S3, gunakan opsi `--s3-key` dan seperti `--s3-bucket` yang ditunjukkan pada perintah contoh berikut. Anda hanya perlu menggunakan `--s3-object-version` parameter untuk objek berversi.

```
aws lambda update-function-code --function-name myFunction \
--s3-bucket amzn-s3-demo-bucket --s3-key myFileName.zip --s3-object-version myObject
 Version
```

Membuat dan memperbarui fungsi dengan file.zip menggunakan API Lambda

Untuk membuat dan memperbarui fungsi menggunakan arsip file.zip, gunakan operasi API berikut:

- CreateFunction
- UpdateFunctionCode

Membuat dan memperbarui fungsi dengan file.zip menggunakan AWS SAM

The AWS Serverless Application Model (AWS SAM) adalah toolkit yang membantu merampingkan proses membangun dan menjalankan aplikasi tanpa server. AWS Anda menentukan sumber daya untuk aplikasi Anda dalam template YAMB atau JSON dan menggunakan antarmuka baris AWS SAM perintah (AWS SAM CLI) untuk membangun, mengemas, dan menyebarkan aplikasi Anda. Saat Anda membuat fungsi Lambda dari AWS SAM template, AWS SAM secara otomatis membuat paket penerapan .zip atau gambar kontainer dengan kode fungsi Anda dan dependensi apa pun yang Anda tentukan. Untuk mempelajari lebih lanjut cara menggunakan AWS SAM untuk membangun dan menerapkan fungsi Lambda, lihat Memulai di Panduan AWS SAMAWS Serverless Application Model Pengembang.

Anda juga dapat menggunakan AWS SAM untuk membuat fungsi Lambda menggunakan arsip file.zip yang ada. Untuk membuat fungsi Lambda menggunakan AWS SAM, Anda dapat menyimpan file.zip di bucket Amazon S3 atau di folder lokal di mesin build Anda. Untuk petunjuk tentang cara mengunggah file ke bucket Amazon S3 menggunakan AWS CLI, lihat Memindahkan objek di AWS CLI Panduan Pengguna.

Dalam AWS SAM template Anda, `AWS::Serverless::Function` sumber daya menentukan fungsi Lambda Anda. Dalam sumber daya ini, atur properti berikut untuk membuat fungsi menggunakan arsip file.zip:

- `PackageType`- diatur ke `Zip`
- `CodeUri`- diatur ke kode fungsi Amazon S3 URI, jalur ke folder lokal, atau objek FunctionCode
- `Runtime`- Setel ke runtime yang Anda pilih

Dengan AWS SAM, jika file.zip Anda lebih besar dari 50MB, Anda tidak perlu mengunggahnya ke bucket Amazon S3 terlebih dahulu. AWS SAM dapat mengunggah paket.zip hingga ukuran maksimum yang diizinkan 250MB (unzip) dari lokasi di mesin build lokal Anda.

Untuk mempelajari selengkapnya tentang penerapan fungsi menggunakan file.zip di AWS SAM, lihat AWS::Serverless: :Fungsi dalam Panduan PengembangAWS SAM .

Membuat dan memperbarui fungsi dengan file.zip menggunakan CloudFormation

Anda dapat menggunakan CloudFormation untuk membuat fungsi Lambda menggunakan arsip file.zip. Untuk membuat fungsi Lambda dari file.zip, Anda harus terlebih dahulu mengunggah file Anda ke bucket Amazon S3. Untuk petunjuk tentang cara mengunggah file ke bucket Amazon S3 menggunakan AWS CLI, lihat Memindahkan objek di AWS CLI Panduan Pengguna.

Dalam CloudFormation template Anda, `AWS::Lambda::Function` sumber daya menentukan fungsi Lambda Anda. Dalam sumber daya ini, atur properti berikut untuk membuat fungsi menggunakan arsip file.zip:

- `PackageType`- Setel ke `Zip`
- `Code`- Masukkan nama bucket Amazon S3 dan nama file.zip di dan bidang `S3Bucket` `S3Key`
- `Runtime`- Setel ke runtime yang Anda pilih

File.zip yang CloudFormation menghasilkan tidak boleh melebihi 4MB. Untuk mempelajari selengkapnya tentang penerapan fungsi menggunakan file.zip di CloudFormation, lihat AWS::Lambda: :Fungsi dalam Panduan Pengguna.CloudFormation

Menyebarkan Fungsi Acara Node.js Lambda dengan gambar kontainer

Ada tiga cara untuk membangun gambar kontainer untuk Fungsi Acara Node.js Lambda:

- Menggunakan sebuah AWS gambar dasar untuk Node.js

  Gambar AWS dasar dimuat sebelumnya dengan runtime bahasa, klien antarmuka runtime untuk mengelola interaksi antara Lambda dan kode fungsi Anda, dan emulator antarmuka runtime untuk pengujian lokal.

- Menggunakan sebuah AWS OS-only gambar dasar

  AWS OS-only image dasar berisi distribusi Amazon Linux dan emulator antarmuka runtime. Gambar-gambar ini biasanya digunakan untuk membuat gambar kontainer untuk bahasa yang dikompilasi, seperti Go dan Rust, dan untuk versi bahasa atau bahasa yang Lambda tidak menyediakan gambar dasar, seperti Node.js 19. Anda juga dapat menggunakan gambar OS-only dasar untuk menerapkan runtime kustom. Untuk membuat gambar kompatibel dengan Lambda, Anda harus menyertakan klien antarmuka runtime untuk Node.js dalam gambar.

- Menggunakan gambar AWS non-dasar

  Anda dapat menggunakan gambar dasar alternatif dari registri kontainer lain, seperti Alpine Linux atau Debian. Anda juga dapat menggunakan gambar kustom yang dibuat oleh organisasi Anda. Untuk membuat gambar kompatibel dengan Lambda, Anda harus menyertakan klien antarmuka runtime untuk Node.js dalam gambar.

> **Tip**
>
> Untuk mengurangi waktu yang dibutuhkan agar fungsi penampung Lambda menjadi aktif, lihat Menggunakan build multi-tahap dalam dokumentasi Docker. Untuk membuat gambar kontainer yang efisien, ikuti Praktik terbaik untuk menulis Dockerfiles.

Halaman ini menjelaskan cara membuat, menguji, dan menyebarkan gambar kontainer untuk Lambda.

Topik

- AWS gambar dasar untuk Node.js
- Menggunakan sebuah AWS gambar dasar untuk Node.js

- Menggunakan gambar dasar alternatif dengan klien antarmuka runtime

AWS gambar dasar untuk Node.js

AWS menyediakan gambar dasar berikut untuk Node.js:

Tanda	Waktu berjalan	Sistem operasi	Dockerfile	penghentian
26	Node.js 26	Amazon Linux 2023	Dockerfile untuk Node.js 26 di GitHub	Tidak dijadwalkan
24	Node.js 24	Amazon Linux 2023	Dockerfile untuk Node.js 24 di GitHub	30 Apr 2028
22	Node.js 22	Amazon Linux 2023	Dockerfile untuk Node.js 22 di GitHub	30 Apr 2027

Repositori Amazon ECR: gallery.ecr. aws/lambda/nodejs

Gambar dasar Node.js 20 dan yang lebih baru didasarkan pada gambar kontainer minimal Amazon Linux 2023. Gambar dasar sebelumnya menggunakan Amazon Linux 2. AL2023 memberikan beberapa keunggulan dibandingkan Amazon Linux 2, termasuk jejak penyebaran yang lebih kecil dan versi pustaka yang diperbarui seperti. `glibc`

AL2023-based gambar menggunakan `microdnf` (symlinked as`dnf`) sebagai manajer paket, bukan`yum`, yang merupakan manajer paket default di Amazon Linux 2. `microdnf`adalah implementasi mandiri dari. `dnf` Untuk daftar paket yang disertakan dalam AL2023-based gambar, lihat kolom Penampung Minimal di Membandingkan paket yang diinstal di Amazon Linux 2023 Container Images. Untuk informasi selengkapnya tentang perbedaan antara AL2023 dan Amazon Linux 2, lihat Memperkenalkan AWS Lambda runtime Amazon Linux 2023 untuk di Blog Komputasi. AWS

> **Note**
>
> Untuk menjalankan AL2023-based gambar secara lokal, termasuk with AWS Serverless Application Model (AWS SAM), Anda harus menggunakan Docker versi 20.10.10 atau yang lebih baru.

Menggunakan sebuah AWS gambar dasar untuk Node.js

Prasyarat

Untuk menyelesaikan langkah-langkah di bagian ini, Anda harus memiliki hal-hal berikut:

- AWS CLI versi 2
- Docker (versi minimum 25.0.0)
- Plugin buildx Docker.
- Node.js

Membuat gambar dari gambar dasar

Untuk membuat gambar kontainer dari sebuah AWS gambar dasar untuk Node.js

1. Buat direktori untuk proyek, dan kemudian beralih ke direktori itu.

   ```
   mkdir example
   cd example
   ```

2. Buat Node.js proyek baru dengan`npm`. Untuk menerima opsi default yang disediakan dalam pengalaman interaktif, tekan`Enter`.

   ```
   npm init
   ```

3. Buat file baru bernama`index.js`. Anda dapat menambahkan kode fungsi contoh berikut ke file untuk pengujian, atau menggunakan kode Anda sendiri.

   Example Penangan CommonJS

   ```
   exports.handler = async (event) => {
       const response = {
   ```

```
        statusCode: 200,
        body: JSON.stringify('Hello from Lambda!'),
    };
    return response;
};
```

4. Jika fungsi Anda bergantung pada pustaka selain AWS SDK untuk JavaScript, gunakan npm untuk menambahkannya ke paket Anda.
5. Buat Dockerfile baru dengan konfigurasi berikut:

   - Mengatur FROM properti ke URI dari gambar dasar.
   - Gunakan perintah COPY untuk menyalin kode fungsi dan dependensi runtime ke{LAMBDA_TASK_ROOT}, variabel lingkungan. Lambda-defined
   - Atur CMD argumen ke penangan fungsi Lambda.

   Perhatikan bahwa contoh Dockerfile tidak menyertakan instruksi USER. Saat Anda menyebarkan gambar kontainer ke Lambda, Lambda secara otomatis mendefinisikan pengguna Linux default dengan izin yang paling tidak memiliki hak istimewa. Ini berbeda dari perilaku Docker standar yang default ke `root` pengguna ketika tidak ada USER instruksi yang diberikan.

   Example Dockerfile

```
FROM public.ecr.aws/lambda/nodejs:22

# Copy function code
COPY index.js ${LAMBDA_TASK_ROOT}

# Set the CMD to your handler (could also be done as a parameter override outside
 of the Dockerfile)
CMD [ "index.handler" ]
```

6. Buat image Docker dengan perintah docker build. Contoh berikut menamai gambar `docker-image` dan memberinya `test` tag. Untuk membuat gambar Anda kompatibel dengan Lambda, Anda harus menggunakan opsi. `--provenance=false`

```
docker buildx build --platform linux/amd64 --provenance=false -t docker-image:test
 .
```

> **Note**
>
> Perintah menentukan `--platform linux/amd64` opsi untuk memastikan bahwa container Anda kompatibel dengan lingkungan eksekusi Lambda terlepas dari arsitektur mesin build Anda. Jika Anda bermaksud membuat fungsi Lambda menggunakan arsitektur set instruksi ARM64, pastikan untuk mengubah perintah untuk menggunakan opsi sebagai gantinya. `--platform linux/arm64`

(Opsional) Uji gambar secara lokal

1. Mulai gambar Docker dengan perintah docker run. Dalam contoh ini, `docker-image` adalah nama gambar dan `test` tag.

   ```
   docker run --platform linux/amd64 -p 9000:8080 docker-image:test
   ```

   Perintah ini menjalankan gambar sebagai wadah dan membuat titik akhir lokal di`localhost:9000/2015-03-31/functions/function/invocations`.

   > **Note**
   >
   > Jika Anda membuat image Docker untuk arsitektur set instruksi ARM64, pastikan untuk menggunakan `--platform linux/arm64` opsi alih-alih. `--platform linux/amd64`

2. Dari jendela terminal baru, posting acara ke titik akhir lokal.

   Linux/macOS

   Di Linux dan macOS, jalankan perintah berikut: `curl`

   ```
   curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
   ```

   Perintah ini memanggil fungsi dengan peristiwa kosong dan mengembalikan respons. Jika Anda menggunakan kode fungsi Anda sendiri daripada kode fungsi sampel, Anda mungkin ingin memanggil fungsi dengan payload JSON. Contoh:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d
 '{"payload":"hello world!"}'
```

PowerShell

Dalam PowerShell, jalankan `Invoke-WebRequest` perintah berikut:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
invocations" -Method Post -Body '{}' -ContentType "application/json"
```

Perintah ini memanggil fungsi dengan peristiwa kosong dan mengembalikan respons. Jika Anda menggunakan kode fungsi Anda sendiri daripada kode fungsi sampel, Anda mungkin ingin memanggil fungsi dengan payload JSON. Contoh:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
invocations" -Method Post -Body '{"payload":"hello world!"}' -ContentType
 "application/json"
```

3. Dapatkan ID kontainer.

```
docker ps
```

4. Gunakan perintah docker kill untuk menghentikan kontainer. Dalam perintah ini, ganti `3766c4ab331c` dengan ID kontainer dari langkah sebelumnya.

```
docker kill 3766c4ab331c
```

Menyebarkan gambar

Untuk mengunggah gambar ke Amazon ECR dan membuat fungsi Lambda

1. Jalankan perintah get-login-password untuk mengautentikasi CLI Docker ke registri Amazon ECR Anda.
   - Tetapkan `--region` nilai ke Wilayah AWS tempat Anda ingin membuat repositori Amazon ECR.
   - Ganti `111122223333` dengan Akun AWS ID Anda.

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --
password-stdin 111122223333.dkr.ecr.us-east-1.amazonaws.com
```

2. Buat repositori di Amazon ECR menggunakan perintah create-repository.

```
aws ecr create-repository --repository-name hello-world --region us-east-1 --image-
scanning-configuration scanOnPush=true --image-tag-mutability MUTABLE
```

> **Note**
>
> Repositori Amazon ECR harus sama Wilayah AWS dengan fungsi Lambda.

Jika berhasil, Anda melihat respons seperti ini:

```
{
    "repository": {
        "repositoryArn": "arn:aws:ecr:us-east-1:111122223333:repository/hello-
world",
        "registryId": "111122223333",
        "repositoryName": "hello-world",
        "repositoryUri": "111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-
world",
        "createdAt": "2023-03-09T10:39:01+00:00",
        "imageTagMutability": "MUTABLE",
        "imageScanningConfiguration": {
            "scanOnPush": true
        },
        "encryptionConfiguration": {
            "encryptionType": "AES256"
        }
    }
}
```

3. Salin `repositoryUri` dari output pada langkah sebelumnya.
4. Jalankan perintah tag docker untuk menandai gambar lokal Anda ke repositori Amazon ECR Anda sebagai versi terbaru. Dalam perintah ini:

- `docker-image:test`adalah nama dan tag gambar Docker Anda. Ini adalah nama gambar dan tag yang Anda tentukan dalam `docker build` perintah.
- Ganti `<ECRrepositoryUri>` dengan `repositoryUri` yang Anda salin. Pastikan untuk menyertakan `:latest` di akhir URI.

```
docker tag docker-image:test <ECRrepositoryUri>:latest
```

Contoh:

```
docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

5. Jalankan perintah docker push untuk menyebarkan gambar lokal Anda ke repositori Amazon ECR. Pastikan untuk menyertakan `:latest` di akhir URI repositori.

```
docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

6. Buat peran eksekusi untuk fungsi tersebut, jika Anda belum memilikinya. Anda memerlukan Nama Sumber Daya Amazon (ARN) dari peran tersebut di langkah berikutnya.
7. Buat fungsi Lambda. Untuk`ImageUri`, tentukan URI repositori dari sebelumnya. Pastikan untuk menyertakan `:latest` di akhir URI.

```
aws lambda create-function \
  --function-name hello-world \
  --package-type Image \
  --code ImageUri=111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
  --role arn:aws:iam::111122223333:role/lambda-ex
```

> **Note**
>
> Anda dapat membuat fungsi menggunakan gambar di AWS akun yang berbeda, selama gambar berada di Wilayah yang sama dengan fungsi Lambda. Untuk informasi selengkapnya, lihat Izin lintas akun Amazon ECR.

8. Memanggil fungsi.

```
aws lambda invoke --function-name hello-world response.json
```

Anda akan melihat tanggapan seperti ini:

```
{
  "ExecutedVersion": "$LATEST",
  "StatusCode": 200
}
```

9. Untuk melihat output dari fungsi, periksa `response.json` file.

Untuk memperbarui kode fungsi, Anda harus membuat gambar lagi, mengunggah gambar baru ke repositori Amazon ECR, dan kemudian menggunakan perintah update-function-code untuk menyebarkan gambar ke fungsi Lambda.

Lambda menyelesaikan tag gambar ke intisari gambar tertentu. Ini berarti bahwa jika Anda mengarahkan tag gambar yang digunakan untuk menyebarkan fungsi ke gambar baru di Amazon ECR, Lambda tidak secara otomatis memperbarui fungsi untuk menggunakan gambar baru.

Untuk menyebarkan gambar baru ke fungsi Lambda yang sama, Anda harus menggunakan perintah update-function-code, meskipun tag gambar di Amazon ECR tetap sama. Dalam contoh berikut, `--publish` opsi membuat versi baru dari fungsi menggunakan gambar kontainer yang diperbarui.

```
aws lambda update-function-code \
  --function-name hello-world \
  --image-uri 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
  --publish
```

Menggunakan gambar dasar alternatif dengan klien antarmuka runtime

Jika Anda menggunakan gambar OS-only dasar atau gambar dasar alternatif, Anda harus menyertakan klien antarmuka runtime dalam gambar Anda. Klien antarmuka runtime memperluasAPI runtime, yang mengelola interaksi antara Lambda dan kode fungsi Anda.

Instal klien antarmuka Node.js runtime menggunakan manajer paket npm:

```
npm install aws-lambda-ric
```

Anda juga dapat mengunduh klien antarmuka Node.js runtime dari GitHub.

Contoh berikut menunjukkan bagaimana membangun image container untuk Node.js menggunakan image AWS non-base. Contoh Dockerfile menggunakan gambar `bookworm` dasar. Dockerfile menyertakan klien antarmuka runtime.

Prasyarat

Untuk menyelesaikan langkah-langkah di bagian ini, Anda harus memiliki hal-hal berikut:

- AWS CLI versi 2
- Docker (versi minimum 25.0.0)
- Plugin buildx Docker.
- Node.js

Membuat gambar dari gambar dasar alternatif

Untuk membuat gambar kontainer dari non-AWS gambar dasar

1. Buat direktori untuk proyek, dan kemudian beralih ke direktori itu.

   ```
   mkdir example
   cd example
   ```

2. Buat Node.js proyek baru dengan`npm`. Untuk menerima opsi default yang disediakan dalam pengalaman interaktif, tekan`Enter`.

   ```
   npm init
   ```

3. Buat file baru bernama`index.js`. Anda dapat menambahkan kode fungsi contoh berikut ke file untuk pengujian, atau menggunakan kode Anda sendiri.

   Example Penangan CommonJS

   ```
   exports.handler = async (event) => {
       const response = {
           statusCode: 200,
           body: JSON.stringify('Hello from Lambda!'),
       };
       return response;
   };
   ```

4. Buat Dockerfile baru. Dockerfile berikut menggunakan gambar `bookworm` dasar bukan gambar AWS dasar. Dockerfile menyertakan klien antarmuka runtime, yang membuat gambar kompatibel dengan Lambda. Dockerfile menggunakan build multi-tahap.

   Tahap pertama membuat image build, yang merupakan Node.js lingkungan standar tempat dependensi fungsi diinstal. Tahap kedua menciptakan gambar yang lebih ramping yang mencakup kode fungsi dan dependensinya. Ini mengurangi ukuran gambar akhir.

   - Atur `FROM` properti ke pengenal gambar dasar.
   - Gunakan `COPY` perintah untuk menyalin kode fungsi dan dependensi runtime.
   - Atur `ENTRYPOINT` ke modul yang Anda inginkan untuk menjalankan wadah Docker saat dimulai. Dalam hal ini, modul adalah klien antarmuka runtime.
   - Atur `CMD` argumen ke penangan fungsi Lambda.

   Perhatikan bahwa contoh Dockerfile tidak menyertakan instruksi USER. Saat Anda menyebarkan gambar kontainer ke Lambda, Lambda secara otomatis mendefinisikan pengguna Linux default dengan izin yang paling tidak memiliki hak istimewa. Ini berbeda dari perilaku Docker standar yang default ke `root` pengguna ketika tidak ada `USER` instruksi yang diberikan.

   Example Dockerfile

```
# Define custom function directory
ARG FUNCTION_DIR="/function"

FROM node:20-bookworm as build-image

# Include global arg in this stage of the build
ARG FUNCTION_DIR

# Install build dependencies
RUN apt-get update && \
    apt-get install -y \
    g++ \
    make \
    cmake \
    unzip \
    libcurl4-openssl-dev

# Copy function code
RUN mkdir -p ${FUNCTION_DIR}
```

```
COPY . ${FUNCTION_DIR}

WORKDIR ${FUNCTION_DIR}

# Install Node.js dependencies
RUN npm install

# Install the runtime interface client
RUN npm install aws-lambda-ric

# Grab a fresh slim copy of the image to reduce the final size
FROM node:20-bookworm-slim

# Required for Node runtimes which use npm@8.6.0+ because
# by default npm writes logs under /home/.npm and Lambda fs is read-only
ENV NPM_CONFIG_CACHE=/tmp/.npm

# Include global arg in this stage of the build
ARG FUNCTION_DIR

# Set working directory to function root directory
WORKDIR ${FUNCTION_DIR}

# Copy in the built dependencies
COPY --from=build-image ${FUNCTION_DIR} ${FUNCTION_DIR}

# Set runtime interface client as default command for the container runtime
ENTRYPOINT ["/usr/local/bin/npx", "aws-lambda-ric"]
# Pass the name of the function handler as an argument to the runtime
CMD ["index.handler"]
```

5. Buat image Docker dengan perintah docker build. Contoh berikut menamai gambar `docker-image` dan memberinya `test` tag. Untuk membuat gambar Anda kompatibel dengan Lambda, Anda harus menggunakan opsi. `--provenance=false`

```
docker buildx build --platform linux/amd64 --provenance=false -t docker-image:test
 .
```

> **Note**
>
> Perintah menentukan `--platform linux/amd64` opsi untuk memastikan bahwa container Anda kompatibel dengan lingkungan eksekusi Lambda terlepas dari arsitektur

mesin build Anda. Jika Anda bermaksud membuat fungsi Lambda menggunakan arsitektur set instruksi ARM64, pastikan untuk mengubah perintah untuk menggunakan opsi sebagai gantinya. `--platform linux/arm64`

(Opsional) Uji gambar secara lokal

Gunakan emulator antarmuka runtime untuk menguji gambar secara lokal. Anda dapat membangun emulator ke dalam gambar Anda atau menggunakan prosedur berikut untuk menginstalnya di mesin lokal Anda.

Untuk menginstal dan menjalankan emulator antarmuka runtime di mesin lokal Anda

1. Dari direktori proyek Anda, jalankan perintah berikut untuk mengunduh emulator antarmuka runtime (arsitektur x86-64) dari GitHub dan menginstalnya di mesin lokal Anda.

   Linux/macOS

   ```
   mkdir -p ~/.aws-lambda-rie && \
       curl -Lo ~/.aws-lambda-rie/aws-lambda-rie https://github.com/aws/aws-lambda-runtime-interface-emulator/releases/latest/download/aws-lambda-rie && \
       chmod +x ~/.aws-lambda-rie/aws-lambda-rie
   ```

   Untuk menginstal emulator arm64, ganti URL GitHub repositori di perintah sebelumnya dengan yang berikut:

   ```
   https://github.com/aws/aws-lambda-runtime-interface-emulator/releases/latest/download/aws-lambda-rie-arm64
   ```

   PowerShell

   ```
   $dirPath = "$HOME\.aws-lambda-rie"
   if (-not (Test-Path $dirPath)) {
       New-Item -Path $dirPath -ItemType Directory
   }

   $downloadLink = "https://github.com/aws/aws-lambda-runtime-interface-emulator/releases/latest/download/aws-lambda-rie"
   $destinationPath = "$HOME\.aws-lambda-rie\aws-lambda-rie"
   Invoke-WebRequest -Uri $downloadLink -OutFile $destinationPath
   ```

Untuk menginstal emulator arm64, ganti `$downloadLink` dengan yang berikut ini:

```
https://github.com/aws/aws-lambda-runtime-interface-emulator/releases/latest/
download/aws-lambda-rie-arm64
```

2. Mulai gambar Docker dengan perintah docker run. Perhatikan hal-hal berikut:

   - `docker-image`adalah nama gambar dan `test` tag.
   - `/usr/local/bin/npx aws-lambda-ric index.handler`adalah `ENTRYPOINT` diikuti oleh `CMD` dari Dockerfile Anda.

   Linux/macOS

   ```
   docker run --platform linux/amd64 -d -v ~/.aws-lambda-rie:/aws-lambda -p
    9000:8080 \
       --entrypoint /aws-lambda/aws-lambda-rie \
       docker-image:test \
           /usr/local/bin/npx aws-lambda-ric index.handler
   ```

   PowerShell

   ```
   docker run --platform linux/amd64 -d -v "$HOME\.aws-lambda-rie:/aws-lambda" -p
    9000:8080 `
   --entrypoint /aws-lambda/aws-lambda-rie `
   docker-image:test `
       /usr/local/bin/npx aws-lambda-ric index.handler
   ```

   Perintah ini menjalankan gambar sebagai wadah dan membuat titik akhir lokal di`localhost:9000/2015-03-31/functions/function/invocations`.

   > **Note**
   >
   > Jika Anda membuat image Docker untuk arsitektur set instruksi ARM64, pastikan untuk menggunakan `--platform linux/arm64` opsi alih-alih. `--platform linux/amd64`

3. Posting acara ke titik akhir lokal.

Linux/macOS

Di Linux dan macOS, jalankan perintah berikut: `curl`

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
```

Perintah ini memanggil fungsi dengan peristiwa kosong dan mengembalikan respons. Jika Anda menggunakan kode fungsi Anda sendiri daripada kode fungsi sampel, Anda mungkin ingin memanggil fungsi dengan payload JSON. Contoh:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d
 '{"payload":"hello world!"}'
```

PowerShell

Dalam PowerShell, jalankan `Invoke-WebRequest` perintah berikut:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
invocations" -Method Post -Body '{}' -ContentType "application/json"
```

Perintah ini memanggil fungsi dengan peristiwa kosong dan mengembalikan respons. Jika Anda menggunakan kode fungsi Anda sendiri daripada kode fungsi sampel, Anda mungkin ingin memanggil fungsi dengan payload JSON. Contoh:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
invocations" -Method Post -Body '{"payload":"hello world!"}' -ContentType
 "application/json"
```

4. Dapatkan ID kontainer.

```
docker ps
```

5. Gunakan perintah docker kill untuk menghentikan kontainer. Dalam perintah ini, ganti `3766c4ab331c` dengan ID kontainer dari langkah sebelumnya.

```
docker kill 3766c4ab331c
```

Menyebarkan gambar

Untuk mengunggah gambar ke Amazon ECR dan membuat fungsi Lambda

1. Jalankan perintah get-login-password untuk mengautentikasi CLI Docker ke registri Amazon ECR Anda.

   - Tetapkan `--region` nilai ke Wilayah AWS tempat Anda ingin membuat repositori Amazon ECR.
   - Ganti `111122223333` dengan Akun AWS ID Anda.

   ```
   aws ecr get-login-password --region us-east-1 | docker login --username AWS --
   password-stdin 111122223333.dkr.ecr.us-east-1.amazonaws.com
   ```

2. Buat repositori di Amazon ECR menggunakan perintah create-repository.

   ```
   aws ecr create-repository --repository-name hello-world --region us-east-1 --image-
   scanning-configuration scanOnPush=true --image-tag-mutability MUTABLE
   ```

   > **Note**
   >
   > Repositori Amazon ECR harus sama Wilayah AWS dengan fungsi Lambda.

   Jika berhasil, Anda melihat respons seperti ini:

   ```
   {
       "repository": {
           "repositoryArn": "arn:aws:ecr:us-east-1:111122223333:repository/hello-
   world",
           "registryId": "111122223333",
           "repositoryName": "hello-world",
           "repositoryUri": "111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-
   world",
           "createdAt": "2023-03-09T10:39:01+00:00",
           "imageTagMutability": "MUTABLE",
           "imageScanningConfiguration": {
               "scanOnPush": true
           },
           "encryptionConfiguration": {
   ```

```
            "encryptionType": "AES256"
        }
    }
}
```

3. Salin `repositoryUri` dari output pada langkah sebelumnya.
4. Jalankan perintah tag docker untuk menandai gambar lokal Anda ke repositori Amazon ECR Anda sebagai versi terbaru. Dalam perintah ini:
   - `docker-image:test`adalah nama dan tag gambar Docker Anda. Ini adalah nama gambar dan tag yang Anda tentukan dalam `docker build` perintah.
   - Ganti <ECRrepositoryUri> dengan `repositoryUri` yang Anda salin. Pastikan untuk menyertakan `:latest` di akhir URI.

```
docker tag docker-image:test <ECRrepositoryUri>:latest
```

   Contoh:

```
docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

5. Jalankan perintah docker push untuk menyebarkan gambar lokal Anda ke repositori Amazon ECR. Pastikan untuk menyertakan `:latest` di akhir URI repositori.

```
docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

6. Buat peran eksekusi untuk fungsi tersebut, jika Anda belum memilikinya. Anda memerlukan Nama Sumber Daya Amazon (ARN) dari peran tersebut di langkah berikutnya.
7. Buat fungsi Lambda. Untuk`ImageUri`, tentukan URI repositori dari sebelumnya. Pastikan untuk menyertakan `:latest` di akhir URI.

```
aws lambda create-function \
  --function-name hello-world \
  --package-type Image \
  --code ImageUri=111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
  --role arn:aws:iam::111122223333:role/lambda-ex
```

> **Note**
>
> Anda dapat membuat fungsi menggunakan gambar di AWS akun yang berbeda, selama gambar berada di Wilayah yang sama dengan fungsi Lambda. Untuk informasi selengkapnya, lihat Izin lintas akun Amazon ECR.

8. Memanggil fungsi.

```
aws lambda invoke --function-name hello-world response.json
```

Anda akan melihat tanggapan seperti ini:

```
{
  "ExecutedVersion": "$LATEST",
  "StatusCode": 200
}
```

9. Untuk melihat output dari fungsi, periksa `response.json` file.

Untuk memperbarui kode fungsi, Anda harus membuat gambar lagi, mengunggah gambar baru ke repositori Amazon ECR, dan kemudian menggunakan perintah update-function-code untuk menyebarkan gambar ke fungsi Lambda.

Lambda menyelesaikan tag gambar ke intisari gambar tertentu. Ini berarti bahwa jika Anda mengarahkan tag gambar yang digunakan untuk menyebarkan fungsi ke gambar baru di Amazon ECR, Lambda tidak secara otomatis memperbarui fungsi untuk menggunakan gambar baru.

Untuk menyebarkan gambar baru ke fungsi Lambda yang sama, Anda harus menggunakan perintah update-function-code, meskipun tag gambar di Amazon ECR tetap sama. Dalam contoh berikut, `--publish` opsi membuat versi baru dari fungsi menggunakan gambar kontainer yang diperbarui.

```
aws lambda update-function-code \
  --function-name hello-world \
  --image-uri 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
  --publish
```

Bekerja dengan lapisan untuk Fungsi Node.js Acara Lambda

Gunakan lapisan Lambda untuk mengemas kode dan dependensi yang ingin Anda gunakan kembali di beberapa fungsi. Lapisan biasanya berisi dependensi pustaka, runtime kustom, atau file konfigurasi. Membuat layer melibatkan tiga langkah umum:

1. Package konten layer Anda. Ini berarti membuat arsip file.zip yang berisi dependensi yang ingin Anda gunakan dalam fungsi Anda.
2. Buat layer di Lambda.
3. Tambahkan layer ke fungsi Anda.

Topik

- Package konten layer Anda
- Buat layer di Lambda
- Tambahkan layer ke fungsi Anda
- Contoh aplikasi

Package konten layer Anda

Untuk membuat layer, bundel paket Anda ke dalam arsip file.zip yang memenuhi persyaratan berikut:

- Bangun layer menggunakan Node.js versi yang sama yang Anda rencanakan untuk digunakan untuk fungsi Lambda. Misalnya, jika Anda membangun layer Anda menggunakan Node.js 24, gunakan runtime Node.js 24 untuk fungsi Anda.
- File.zip layer Anda harus menggunakan salah satu struktur direktori ini:
  - `nodejs/node_modules`
  - `nodejs/node`*X*`/node_modules`(di *X* mana Node.js versi Anda, misalnya`node22`)

  Untuk informasi selengkapnya, lihat Jalur lapisan untuk setiap runtime Lambda.
- Paket di lapisan Anda harus kompatibel dengan Linux. Fungsi Lambda berjalan di Amazon Linux.

Anda dapat membuat lapisan yang berisi Node.js pustaka pihak ketiga yang diinstal dengan `npm` (seperti `axios` atau`lodash`) atau JavaScript modul Anda sendiri.

Third-party dependensi

Untuk membuat layer menggunakan paket npm

1. Buat struktur direktori yang diperlukan dan instal paket langsung ke dalamnya:

```
mkdir -p nodejs
npm install --prefix nodejs lodash axios
```

Perintah ini menginstal paket langsung ke `nodejs/node_modules` direktori, yang merupakan struktur yang dibutuhkan Lambda.

> **Note**
>
> Untuk paket dengan dependensi asli atau komponen biner (seperti sharp atau bcrypt), pastikan bahwa mereka kompatibel dengan lingkungan Linux Lambda dan arsitektur fungsi Anda. Anda mungkin perlu menggunakan `--platform` bendera:
>
> ```
> npm install --prefix nodejs --platform=linux --arch=x64 sharp
> ```
>
> Untuk dependensi asli yang lebih kompleks, Anda mungkin perlu mengompilasinya di lingkungan Linux yang cocok dengan runtime Lambda. Anda dapat menggunakan Docker untuk tujuan ini.

2. Zip konten lapisan:

Linux/macOS

```
zip -r layer.zip nodejs/
```

PowerShell

```
Compress-Archive -Path .\nodejs -DestinationPath .\layer.zip
```

Struktur direktori file.zip Anda akan terlihat seperti ini:

```
nodejs/
### package.json
```

```
### package-lock.json
### node_modules/
    ### lodash/
    ### axios/
    ### (dependencies of the other packages)
```

> **Note**
>
> - Pastikan file.zip Anda menyertakan `nodejs` direktori di tingkat root dengan `node_modules` di dalamnya. Struktur ini memastikan bahwa Lambda dapat menemukan dan mengimpor paket Anda.
> - `package-lock.json`File `package.json` dan dalam `nodejs/` direktori digunakan oleh npm untuk manajemen ketergantungan tetapi tidak diperlukan oleh Lambda untuk fungsionalitas lapisan. Setiap paket yang diinstal sudah berisi `package.json` file sendiri yang mendefinisikan bagaimana Lambda mengimpor paket.

JavaScript Modul kustom

Untuk membuat layer menggunakan kode Anda sendiri

1. Buat struktur direktori yang diperlukan untuk layer Anda:

   ```
   mkdir -p nodejs/node_modules/validator
   cd nodejs/node_modules/validator
   ```

2. Buat `package.json` file untuk modul kustom Anda untuk menentukan bagaimana seharusnya diimpor:

   Example nodejs/node_modules/validator/package.json

   ```
   {
     "name": "validator",
     "version": "1.0.0",
     "type": "module",
     "main": "index.mjs"
   }
   ```

3. Buat file JavaScript modul Anda:

Example nodejs/node_modules/validator/index.mjs

```
export function validateOrder(orderData) {
  // Validates an order and returns formatted data
  const requiredFields = ['productId', 'quantity'];

  // Check required fields
  const missingFields = requiredFields.filter(field => !(field in orderData));
  if (missingFields.length > 0) {
    throw new Error(`Missing required fields: ${missingFields.join(', ')}`);
  }

  // Validate quantity
  const quantity = orderData.quantity;
  if (!Number.isInteger(quantity) || quantity < 1) {
    throw new Error('Quantity must be a positive integer');
  }

  // Format and return the validated data
  return {
    productId: String(orderData.productId),
    quantity: quantity,
    shippingPriority: orderData.priority || 'standard'
  };
}

export function formatResponse(statusCode, body) {
  // Formats the API response
  return {
    statusCode: statusCode,
    body: JSON.stringify(body)
  };
}
```

4. Zip konten lapisan:

Linux/macOS

```
zip -r layer.zip nodejs/
```

PowerShell

```
Compress-Archive -Path .\nodejs -DestinationPath .\layer.zip
```

Struktur direktori file.zip Anda akan terlihat seperti ini:

```
nodejs/
### node_modules/
    ### validator/
        ### package.json
        ### index.mjs
```

5. Dalam fungsi Anda, impor dan gunakan modul. Contoh:

```
import { validateOrder, formatResponse } from 'validator';

export const handler = async (event) => {
  try {
    // Parse the order data from the event body
    const orderData = JSON.parse(event.body || '{}');

    // Validate and format the order
    const validatedOrder = validateOrder(orderData);

    return formatResponse(200, {
      message: 'Order validated successfully',
      order: validatedOrder
    });
  } catch (error) {
    if (error instanceof Error && error.message.includes('Missing required
 fields')) {
      return formatResponse(400, {
        error: error.message
      });
    }

    return formatResponse(500, {
      error: 'Internal server error'
    });
  }
```

```
};
```

Anda dapat menggunakan peristiwa pengujian berikut untuk menjalankan fungsi:

```
{
    "body": "{\"productId\": \"ABC123\", \"quantity\": 2, \"priority\": \"express
\"}"
}
```

Tanggapan yang diharapkan:

```
{
  "statusCode": 200,
  "body": "{\"message\":\"Order validated successfully\",\"order\":{\"productId\":
\"ABC123\",\"quantity\":2,\"shippingPriority\":\"express\"}}"
}
```

Buat layer di Lambda

Anda dapat mempublikasikan layer Anda menggunakan konsol AWS CLI atau Lambda.

AWS CLI

Jalankan AWS CLI perintah publish-layer-version untuk membuat layer Lambda:

```
aws lambda publish-layer-version --layer-name my-layer --zip-file fileb://layer.zip
 --compatible-runtimes nodejs24.x
```

Parameter runtime yang kompatibel adalah opsional. Saat ditentukan, Lambda menggunakan parameter ini untuk memfilter lapisan di konsol Lambda.

Console

Untuk membuat lapisan (konsol)

1. Buka halaman Fungsi di konsol Lambda.
2. Pilih Buat lapisan.
3. Pilih Unggah file.zip, lalu unggah arsip.zip yang Anda buat sebelumnya.
4. (Opsional) Untuk runtime yang kompatibel, pilih Node.js runtime yang sesuai dengan Node.js versi yang Anda gunakan untuk membangun layer Anda.

5. Pilih Buat.

Tambahkan layer ke fungsi Anda

AWS CLI

Untuk melampirkan layer ke fungsi Anda, jalankan perintah AWS CLI update-function-configuration. Untuk `--layers` parameter, gunakan lapisan ARN. ARN harus menentukan versi (misalnya,`arn:aws:lambda:us-east-1:123456789012:layer:my-layer:1`). Untuk informasi selengkapnya, lihat Versi lapisan dan lapisan.

```
aws lambda update-function-configuration --function-name my-function --cli-binary-format raw-in-base64-out --layers "arn:aws:lambda:us-east-1:123456789012:layer:my-layer:1"
```

cli-binary-formatOpsi ini diperlukan jika Anda menggunakan AWS CLI versi 2. Untuk menjadikan ini pengaturan default, jalankan`aws configure set cli-binary-format raw-in-base64-out`. Untuk informasi selengkapnya, lihat opsi baris perintah global yang AWS CLI didukung di Panduan AWS Command Line Interface Pengguna untuk Versi 2.

Console

Untuk menambahkan lapisan ke fungsi

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih fungsinya.
3. Gulir ke bawah ke bagian Layers, lalu pilih Add a layer.
4. Di bawah Pilih layer, pilih Custom layers, dan kemudian pilih layer Anda.

> **Note**
>
> Jika Anda tidak menambahkan runtime yang kompatibel saat membuat layer, layer Anda tidak akan tercantum di sini. Anda dapat menentukan lapisan ARN sebagai gantinya.

5. Pilih Tambahkan.

Contoh aplikasi

Untuk contoh lebih lanjut tentang cara menggunakan layer Lambda, lihat contoh aplikasi layer-nodejs di repositori Panduan Pengembang. AWS Lambda GitHub Aplikasi ini mencakup lapisan yang berisi pustaka lodash. Setelah membuat layer, Anda dapat menerapkan dan memanggil fungsi yang sesuai untuk mengonfirmasi bahwa lapisan berfungsi seperti yang diharapkan.

Menggunakan objek konteks Lambda untuk mengambil informasi Fungsi Acara Node.js

Saat Lambda menjalankan fungsi Anda, ia meneruskan objek konteks ke handler. Objek ini menyediakan metode dan properti yang memberikan informasi tentang lingkungan invokasi, fungsi, dan eksekusi.

Metode konteks

- `getRemainingTimeInMillis()` – Mengembalikan jumlah milidetik yang tersisa sebelum waktu eksekusi habis.

Properti konteks

- `functionName` – Nama fungsi Lambda.
- `functionVersion` – Versi fungsi.
- `invokedFunctionArn` – Amazon Resource Name (ARN) yang digunakan untuk memicu fungsi. Menunjukkan jika pemicu menyebutkan nomor versi atau alias.
- `memoryLimitInMB` – Jumlah memori yang dialokasikan untuk fungsi tersebut.
- `awsRequestId` – Pengidentifikasi permintaan invokasi.
- `logGroupName` – Grup log untuk fungsi.
- `logStreamName` – Aliran log untuk instans fungsi.
- `identity` – (aplikasi seluler) Informasi tentang identitas Amazon Cognito yang mengesahkan permintaan.
  - `cognitoIdentityId` – Identitas Amazon Cognito terautentikasi.
  - `cognitoIdentityPoolId` – Kumpulan identitas Amazon Cognito yang mengesahkan invokasi.
- `clientContext` – (aplikasi seluler) Konteks klien yang disediakan untuk Lambda oleh aplikasi klien.
  - `client.installation_id`
  - `client.app_title`
  - `client.app_version_name`
  - `client.app_version_code`
  - `client.app_package_name`
  - `env.platform_version`

  - `env.platform`
  - `env.make`
  - `env.model`
  - `env.locale`
  - `custom` – Nilai kustom yang ditetapkan oleh aplikasi klien.
- `callbackWaitsForEmptyEventLoop`— Secara default (`true`), saat menggunakan penangan fungsi berbasis callback, Lambda menunggu loop peristiwa kosong setelah panggilan balik berjalan sebelum mengakhiri pemanggilan fungsi. Setel `false` untuk mengirim respons dan mengakhiri pemanggilan segera setelah panggilan balik berjalan alih-alih menunggu loop acara kosong. Acara luar biasa terus berjalan selama doa berikutnya. Perhatikan bahwa Lambda mendukung penangan fungsi berbasis callback hanya untuk Node.js 22 runtime dan sebelumnya.

Contoh fungsi berikut mencatat informasi konteks dan mengembalikan lokasi log.

Example index.js file

```
exports.handler = async function(event, context) {
  console.log('Remaining time: ', context.getRemainingTimeInMillis())
  console.log('Function name: ', context.functionName)
  return context.logStreamName
}
```

Log dan pantau Node.js Fungsi Acara Lambda

AWS Lambda secara otomatis memonitor Fungsi Peristiwa Lambda atas nama Anda dan mengirim log ke Amazon. CloudWatch Fungsi Anda dilengkapi dengan grup CloudWatch log Log dan aliran log untuk setiap instance fungsi Anda. Lingkungan runtime Lambda mengirimkan detail tentang setiap invokasi ke pengaliran log, dan menyampaikan log serta output lain dari kode fungsi Anda. Untuk informasi selengkapnya, lihat Mengirim log Fungsi Peristiwa Lambda ke Log CloudWatch.

Halaman ini menjelaskan cara menghasilkan keluaran log dari kode fungsi Lambda Anda, dan mengakses log menggunakan AWS Command Line Interface, konsol Lambda, atau konsol. CloudWatch

Bagian-bagian

- Membuat fungsi yang mengembalikan log
- Menggunakan kontrol logging lanjutan Lambda dengan Node.js
- Melihat log di konsol Lambda
- Melihat log di CloudWatch konsol
- Melihat log menggunakan AWS Command Line Interface (AWS CLI)
- Menghapus log

Membuat fungsi yang mengembalikan log

Untuk menghasilkan log dari kode fungsi, Anda dapat menggunakan metode di objek konsol, atau pustaka pencatatan yang menulis ke `stdout` atau `stderr`. Contoh berikut mencatat nilai variabel lingkungan dan objek peristiwa.

> **Note**
>
> Kami menyarankan Anda menggunakan teknik seperti validasi input dan pengkodean output saat mencatat input. Jika Anda mencatat data input secara langsung, penyerang mungkin dapat menggunakan kode Anda untuk membuat gangguan sulit dideteksi, memalsukan entri log, atau melewati monitor log. Untuk informasi selengkapnya, lihat Netralisasi Output yang Tidak Tepat untuk Log dalam Pencacahan Kelemahan Umum.

Example index.js file – Pencatatan

```
exports.handler = async function(event, context) {
  console.log("ENVIRONMENT VARIABLES\n" + JSON.stringify(process.env, null, 2))
  console.info("EVENT\n" + JSON.stringify(event, null, 2))
  console.warn("Event not processed.")
  return context.logStreamName
}
```

Example format log

```
START RequestId: c793869b-ee49-115b-a5b6-4fd21e8dedac Version: $LATEST
2019-06-07T19:11:20.562Z c793869b-ee49-115b-a5b6-4fd21e8dedac INFO ENVIRONMENT
 VARIABLES
{
  "AWS_LAMBDA_FUNCTION_VERSION": "$LATEST",
  "AWS_LAMBDA_LOG_GROUP_NAME": "/aws/lambda/my-function",
  "AWS_LAMBDA_LOG_STREAM_NAME": "2019/06/07/[$LATEST]e6f4a0c4241adcd70c262d34c0bbc85c",
  "AWS_EXECUTION_ENV": "AWS_Lambda_nodejs12.x",
  "AWS_LAMBDA_FUNCTION_NAME": "my-function",
  "PATH": "/var/lang/bin:/usr/local/bin:/usr/bin/:/bin:/opt/bin",
  "NODE_PATH": "/opt/nodejs/node10/node_modules:/opt/nodejs/node_modules:/var/runtime/
node_modules",
  ...
}
2019-06-07T19:11:20.563Z c793869b-ee49-115b-a5b6-4fd21e8dedac INFO EVENT
{
  "key": "value"
}
2019-06-07T19:11:20.564Z c793869b-ee49-115b-a5b6-4fd21e8dedac WARN Event not processed.
END RequestId: c793869b-ee49-115b-a5b6-4fd21e8dedac
REPORT RequestId: c793869b-ee49-115b-a5b6-4fd21e8dedac Duration: 128.83 ms Billed
 Duration: 296 ms Memory Size: 128 MB Max Memory Used: 74 MB Init Duration: 166.62 ms
 XRAY TraceId: 1-5d9d007f-0a8c7fd02xmpl480aed55ef0 SegmentId: 3d752xmpl1bbe37e Sampled:
 true
```

Node.js Runtime mencatat `REPORT` baris`START`,`END`, dan untuk setiap pemanggilan. Ini menambahkan stempel waktu, ID permintaan, dan tingkat log ke setiap entri yang dicatat oleh fungsi. Baris laporan memberikan perincian berikut.

Laporkan bidang data baris

- RequestId – ID permintaan unik untuk invokasi.
- Durasi – Jumlah waktu yang digunakan oleh metode handler fungsi Anda gunakan untuk memproses peristiwa.
- Durasi yang Ditagih – Jumlah waktu yang ditagihkan untuk invokasi.
- Ukuran Memori – Jumlah memori yang dialokasikan untuk fungsi.
- Memori Maks yang Digunakan – Jumlah memori yang digunakan oleh fungsi. Saat pemanggilan berbagi lingkungan eksekusi, Lambda melaporkan memori maksimum yang digunakan di semua pemanggilan. Perilaku ini dapat menghasilkan nilai yang dilaporkan lebih tinggi dari yang diharapkan.
- Durasi Init – Untuk permintaan pertama yang dilayani, lama waktu yang diperlukan runtime untuk memuat fungsi dan menjalankan kode di luar metode handler.
- XRAY TraceId — Untuk permintaan yang dilacak, ID AWS X-Ray jejak.
- SegmentId— Untuk permintaan yang dilacak, ID X-Ray segmen.
- Diambil Sampel – Untuk permintaan yang dilacak, hasil pengambilan sampel.

Anda dapat melihat log di konsol Lambda, di konsol CloudWatch Log, atau dari baris perintah.

Menggunakan kontrol logging lanjutan Lambda dengan Node.js

Untuk memberi Anda kontrol lebih besar atas cara log fungsi Anda ditangkap, diproses, dan digunakan, Anda dapat mengonfigurasi opsi pencatatan berikut untuk Node.js runtime yang didukung:

- Format log - pilih antara teks biasa dan format JSON terstruktur untuk log fungsi Anda
- Tingkat log - untuk log dalam format JSON, pilih tingkat detail log yang dikirim Lambda ke CloudWatch Amazon, seperti ERROR, DEBUG, atau INFO
- Grup log - pilih grup CloudWatch log fungsi Anda mengirim log ke

Untuk informasi selengkapnya tentang opsi pencatatan ini, dan petunjuk tentang cara mengonfigurasi fungsi Anda untuk menggunakannya, lihatthe section called “Mengkonfigurasi kontrol logging lanjutan untuk Lambda Event Functions”.

Untuk menggunakan format log dan opsi tingkat log dengan fungsi Node.js Lambda Anda, lihat panduan di bagian berikut.

Menggunakan log JSON terstruktur dengan Node.js

Jika Anda memilih JSON untuk format log fungsi Anda, Lambda mengirimkan keluaran log menggunakan metode `console.trace` konsol`console.debug`,,,, `console.log` `console.info`, `console.error` dan `console.warn` CloudWatch ke sebagai JSON terstruktur. Setiap objek log JSON berisi setidaknya empat pasangan nilai kunci dengan kunci berikut:

- `"timestamp"`- waktu pesan log dihasilkan
- `"level"`- tingkat log yang ditetapkan untuk pesan
- `"message"`- isi pesan log
- `"requestId"`- ID permintaan unik untuk pemanggilan fungsi

Bergantung pada metode logging yang digunakan fungsi Anda, objek JSON ini mungkin juga berisi pasangan kunci tambahan. Misalnya, jika fungsi Anda menggunakan `console` metode untuk mencatat objek kesalahan menggunakan beberapa argumen, objek JSON akan berisi pasangan nilai kunci tambahan dengan kunci`errorMessage`,`errorType`, dan`stackTrace`.

Jika kode Anda sudah menggunakan pustaka logging lain, seperti Powertools for AWS Lambda, untuk menghasilkan log terstruktur JSON, Anda tidak perlu membuat perubahan apa pun. Lambda tidak menyandikan dua kali log apa pun yang sudah dikodekan JSON, sehingga log aplikasi fungsi Anda terus ditangkap seperti sebelumnya.

Untuk informasi selengkapnya tentang penggunaan paket Powertools for AWS Lambda logging guna membuat log terstruktur JSON di Node.js runtime, lihat. the section called “Pencatatan log”

Contoh keluaran log yang diformat JSON

Contoh berikut menunjukkan bagaimana berbagai keluaran log yang dihasilkan menggunakan `console` metode dengan argumen tunggal dan ganda ditangkap di CloudWatch Log saat Anda menyetel format log fungsi Anda ke JSON.

Contoh pertama menggunakan `console.error` metode untuk menampilkan string sederhana.

Example Node.js kode logging

```
export const handler = async (event) => {
  console.error("This is a warning message");
  ...
```

```
}
```

Example Catatan log JSON

```
{
    "timestamp":"2025-11-01T00:21:51.358Z",
    "level":"ERROR",
    "message":"This is a warning message",
    "requestId":"93f25699-2cbf-4976-8f94-336a0aa98c6f"
}
```

Anda juga dapat menampilkan pesan log terstruktur yang lebih kompleks menggunakan argumen tunggal atau ganda dengan `console` metode. Dalam contoh berikutnya, Anda gunakan `console.log` untuk menampilkan dua pasangan nilai kunci menggunakan argumen tunggal. Perhatikan bahwa `"message"` bidang dalam objek JSON yang dikirim Lambda CloudWatch ke Log tidak dirangkai.

Example Node.js kode logging

```
export const handler = async (event) => {
  console.log({data: 12.3, flag: false});
  ...
}
```

Example Catatan log JSON

```
{
    "timestamp": "2025-12-08T23:21:04.664Z",
    "level": "INFO",
    "requestId": "405a4537-9226-4216-ac59-64381ec8654a",
    "message": {
        "data": 12.3,
        "flag": false
    }
}
```

Dalam contoh berikutnya, Anda kembali menggunakan `console.log` metode untuk membuat output log. Kali ini, metode ini mengambil dua argumen, peta yang berisi dua pasangan nilai kunci dan string pengidentifikasi. Perhatikan bahwa dalam kasus ini, karena Anda telah memberikan dua argumen, Lambda membuat stringifikasi bidang. `"message"`

Example Node.js kode logging

```
export const handler = async (event) => {
  console.log('Some object - ', {data: 12.3, flag: false});
  ...
}
```

Example Catatan log JSON

```
{
    "timestamp": "2025-12-08T23:21:04.664Z",
    "level": "INFO",
    "requestId": "405a4537-9226-4216-ac59-64381ec8654a",
    "message": "Some object -  { data: 12.3, flag: false }"
}
```

Lambda menetapkan output yang dihasilkan menggunakan INFO tingkat log`console.log`.

Contoh terakhir menunjukkan bagaimana objek kesalahan dapat output ke CloudWatch Log menggunakan `console` metode. Perhatikan bahwa ketika Anda mencatat objek kesalahan menggunakan beberapa argumen, Lambda menambahkan bidang`errorMessage`,`errorType`, dan `stackTrace` ke output log.

Example Node.js kode logging

```
export const handler = async (event) => {
  let e1 = new ReferenceError("some reference error");
  let e2 = new SyntaxError("some syntax error");
  console.log(e1);
  console.log("errors logged - ", e1, e2);
};
```

Example Catatan log JSON

```
{
    "timestamp": "2025-12-08T23:21:04.632Z",
    "level": "INFO",
    "requestId": "405a4537-9226-4216-ac59-64381ec8654a",
    "message": {
        "errorType": "ReferenceError",
```

```
        "errorMessage": "some reference error",
        "stackTrace": [
            "ReferenceError: some reference error",
            "    at Runtime.handler (file:///var/task/index.mjs:3:12)",
            "    at Runtime.handleOnceNonStreaming (file:///var/runtime/
index.mjs:1173:29)"
        ]
    }
}

{
    "timestamp": "2025-12-08T23:21:04.646Z",
    "level": "INFO",
    "requestId": "405a4537-9226-4216-ac59-64381ec8654a",
    "message": "errors logged -  ReferenceError: some reference error
\n    at Runtime.handler (file:///var/task/index.mjs:3:12)\n    at
 Runtime.handleOnceNonStreaming
    (file:///var/runtime/index.mjs:1173:29) SyntaxError: some syntax
 error\n    at Runtime.handler (file:///var/task/index.mjs:4:12)\n    at
 Runtime.handleOnceNonStreaming
    (file:///var/runtime/index.mjs:1173:29)",
    "errorType": "ReferenceError",
    "errorMessage": "some reference error",
    "stackTrace": [
        "ReferenceError: some reference error",
        "    at Runtime.handler (file:///var/task/index.mjs:3:12)",
        "    at Runtime.handleOnceNonStreaming (file:///var/runtime/index.mjs:1173:29)"
    ]
}
```

Saat mencatat beberapa jenis kesalahan, bidang tambahan`errorMessage`,`errorType`, dan `stackTrace` diekstraksi dari jenis kesalahan pertama yang diberikan ke `console` metode.

Menggunakan pustaka klien format metrik tertanam (EMF) dengan log JSON terstruktur

AWS menyediakan pustaka klien bersumber terbuka Node.js yang dapat Anda gunakan untuk membuat log format metrik tertanam (EMF). Jika Anda memiliki fungsi yang ada yang menggunakan pustaka ini dan Anda mengubah format log fungsi Anda ke JSON, CloudWatch mungkin tidak lagi mengenali metrik yang dipancarkan oleh kode Anda.

Jika kode Anda saat ini memancarkan log EMF secara langsung menggunakan `console.log` atau dengan menggunakan Powertools for AWS Lambda (TypeScript), CloudWatch juga tidak dapat mengurai ini jika Anda mengubah format log fungsi Anda menjadi JSON.

> **Important**
>
> Untuk memastikan bahwa log EMF fungsi Anda terus diurai dengan benar CloudWatch, perbarui EMF dan Powertools Anda untuk AWS Lambda pustaka ke versi terbaru. Jika beralih ke format log JSON, kami juga menyarankan Anda melakukan pengujian untuk memastikan kompatibilitas dengan metrik tertanam fungsi Anda. Jika kode Anda memancarkan log EMF secara langsung menggunakan`console.log`, ubah kode Anda untuk menampilkan metrik tersebut secara langsung `stdout` seperti yang ditunjukkan pada contoh kode berikut.

Example **`kode yang memancarkan metrik tertanam ke stdout`**

```
process.stdout.write(JSON.stringify(
    {
        "_aws": {
            "Timestamp": Date.now(),
            "CloudWatchMetrics": [{
                "Namespace": "lambda-function-metrics",
                "Dimensions": [["functionVersion"]],
                "Metrics": [{
                    "Name": "time",
                    "Unit": "Milliseconds",
                    "StorageResolution": 60
                }]
            }]
        },
        "functionVersion": "$LATEST",
        "time": 100,
        "requestId": context.awsRequestId
    }
) + "\n")
```

Menggunakan penyaringan tingkat log dengan Node.js

AWS Lambda Untuk memfilter log aplikasi Anda sesuai dengan tingkat lognya, fungsi Anda harus menggunakan log berformat JSON. Anda dapat mencapai ini dengan dua cara:

- Buat output log menggunakan metode konsol standar dan konfigurasikan fungsi Anda untuk menggunakan pemformatan log JSON. AWS Lambda kemudian memfilter output log Anda menggunakan pasangan nilai kunci “level” di objek JSON yang dijelaskan dalam. the section called

"Menggunakan log JSON terstruktur dengan Node.js" Untuk mempelajari cara mengonfigurasi format log fungsi Anda, lihatthe section called "Mengkonfigurasi kontrol logging lanjutan untuk Lambda Event Functions".

- Gunakan pustaka atau metode logging lain untuk membuat log terstruktur JSON dalam kode Anda yang menyertakan pasangan nilai kunci "level" yang menentukan tingkat keluaran log. Misalnya, Anda dapat menggunakan Powertools AWS Lambda untuk menghasilkan output log terstruktur JSON dari kode Anda. Lihat the section called "Pencatatan log" untuk mempelajari lebih lanjut tentang menggunakan Powertools dengan Node.js runtime.

  Agar Lambda dapat memfilter log fungsi Anda, Anda juga harus menyertakan pasangan nilai `"timestamp"` kunci dalam keluaran log JSON Anda. Waktu harus ditentukan dalam format stempel waktu RFC 3339 yang valid. Jika Anda tidak menyediakan stempel waktu yang valid, Lambda menetapkan log INFO level dan menambahkan stempel waktu untuk Anda.

Ketika Anda mengonfigurasi fungsi Anda untuk menggunakan pemfilteran tingkat log, Anda memilih tingkat log yang ingin Anda kirim AWS Lambda ke CloudWatch Log dari opsi berikut:

Tingkat log	Penggunaan standar
TRACE (paling detail)	Informasi paling halus yang digunakan untuk melacak jalur eksekusi kode Anda
DEBUG	Informasi terperinci untuk debugging sistem
INFO	Pesan yang merekam operasi normal fungsi Anda
WARN	Pesan tentang potensi kesalahan yang dapat menyebabkan perilaku tak terduga jika tidak ditangani
Kesalahan	Pesan tentang masalah yang mencegah kode berfungsi seperti yang diharapkan
FATAL (paling detail)	Pesan tentang kesalahan serius yang menyebabkan aplikasi berhenti berfungsi

Lambda mengirimkan log dari level yang dipilih dan lebih rendah ke. CloudWatch Misalnya, jika Anda mengonfigurasi level log WARN, Lambda mengirimkan log yang sesuai dengan level WARN, ERROR, dan FATAL.

Melihat log di konsol Lambda

Anda dapat menggunakan konsol Lambda untuk melihat keluaran log setelah Anda menjalankan Fungsi Peristiwa Lambda.

Jika kode Anda dapat diuji dari editor Kode tertanam, Anda menemukan log dalam hasil eksekusi. Saat Anda menggunakan fitur pengujian konsol untuk menjalankan fungsi, Anda menemukan Keluaran Log di bagian Detail.

Melihat log di CloudWatch konsol

Anda dapat menggunakan CloudWatch konsol Amazon untuk melihat log untuk semua pemanggilan Fungsi Peristiwa Lambda.

Untuk melihat log di CloudWatch konsol

1. Buka halaman Grup log di CloudWatch konsol.
2. Pilih grup log untuk fungsi Anda (/aws/lambda/***your-function-name***).
3. Pilih pengaliran log.

Setiap aliran log sesuai dengan instans fungsi Anda. Aliran log muncul saat Anda memperbarui fungsi Lambda, dan saat instance tambahan dibuat untuk menangani pemanggilan bersamaan. Untuk menemukan log untuk pemanggilan tertentu, kami sarankan untuk menginstrumentasi fungsi Anda dengan. AWS X-Ray X-Ray mencatat detail tentang permintaan dan aliran log dalam jejak.

Melihat log menggunakan AWS Command Line Interface (AWS CLI)

AWS CLI Ini adalah alat sumber terbuka yang dapat Anda gunakan untuk berinteraksi dengan AWS layanan menggunakan perintah di shell baris perintah Anda. Untuk menyelesaikan langkah-langkah di bagian ini, Anda harus memiliki AWS CLI versi 2.

Anda dapat menggunakan AWS CLI untuk mengambil log untuk invokasi menggunakan opsi perintah `--log-type`. Respons berisi bidang `LogResult` yang memuat hingga 4 KB log berkode base64 dari invokasi.

Example mengambil ID log

Contoh berikut menunjukkan cara mengambil ID log dari `LogResult` untuk fungsi bernama `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

Anda akan melihat output berikut:

```
{
    "StatusCode": 200,
    "LogResult":
 "U1RBUlQgUmVxdWVzdElkOiA4N2QwNDRiOC1mMTU0LTExZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "ExecutedVersion": "$LATEST"
}
```

Example mendekode log

Pada prompt perintah yang sama, gunakan utilitas `base64` untuk mendekodekan log. Contoh berikut menunjukkan cara mengambil log berkode base64 untuk `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail \
--query 'LogResult' --output text --cli-binary-format raw-in-base64-out | base64 --
decode
```

cli-binary-formatOpsi ini diperlukan jika Anda menggunakan AWS CLI versi 2. Untuk menjadikan ini pengaturan default, jalankan`aws configure set cli-binary-format raw-in-base64-out`. Untuk informasi selengkapnya, lihat opsi baris perintah global yang AWS CLI didukung di Panduan AWS Command Line Interface Pengguna untuk Versi 2.

Anda akan melihat output berikut:

```
START RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Version: $LATEST
"AWS_SESSION_TOKEN": "AgoJb3JpZ2luX2VjELj...", "_X_AMZN_TRACE_ID": "Root=1-5d02e5ca-
f5792818b6fe8368e5b51d50;Parent=191db58857df8395;Sampled=0"",ask/lib:/opt/lib",
END RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8
REPORT RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8  Duration: 79.67 ms      Billed
 Duration: 80 ms         Memory Size: 128 MB     Max Memory Used: 73 MB
```

Utilitas `base64` tersedia di Linux, macOS, dan Ubuntu pada Windows. Pengguna macOS mungkin harus menggunakan `base64 -D`.

Example Skrip get-logs.sh

Pada prompt perintah yang sama, gunakan script berikut untuk mengunduh lima peristiwa log terakhir. Skrip menggunakan `sed` untuk menghapus kutipan dari file output, dan akan tidur selama 15 detik untuk memberikan waktu agar log tersedia. Output mencakup respons dari Lambda dan output dari perintah `get-log-events`.

Salin konten dari contoh kode berikut dan simpan dalam direktori proyek Lambda Anda sebagai `get-logs.sh`.

cli-binary-formatOpsi ini diperlukan jika Anda menggunakan AWS CLI versi 2. Untuk menjadikan ini pengaturan default, jalankan`aws configure set cli-binary-format raw-in-base64-out`. Untuk informasi selengkapnya, lihat opsi baris perintah global yang AWS CLI didukung di Panduan AWS Command Line Interface Pengguna untuk Versi 2.

```
#!/bin/bash
aws lambda invoke --function-name my-function --cli-binary-format raw-in-base64-out --
payload '{"key": "value"}' out
sed -i'' -e 's/"//g' out
sleep 15
aws logs get-log-events --log-group-name /aws/lambda/my-function --log-stream-
name stream1 --limit 5
```

Example macOS dan Linux (khusus)

Dalam prompt perintah yang sama, pengguna macOS dan Linux mungkin perlu menjalankan perintah berikut untuk memastikan skrip dapat dieksekusi.

```
chmod -R 755 get-logs.sh
```

Example mengambil lima log acara terakhir

Pada prompt perintah yang sama, gunakan skrip berikut untuk mendapatkan lima log acara terakhir.

```
./get-logs.sh
```

Anda akan melihat output berikut:

```
{
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
```

```
}
{
    "events": [
        {
            "timestamp": 1559763003171,
            "message": "START RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf Version:
 $LATEST\n",
            "ingestionTime": 1559763003309
        },
        {
            "timestamp": 1559763003173,
            "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tENVIRONMENT VARIABLES\r{\r  \"AWS_LAMBDA_FUNCTION_VERSION\": \"$LATEST\",
\r ...",
            "ingestionTime": 1559763018353
        },
        {
            "timestamp": 1559763003173,
            "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tEVENT\r{\r  \"key\": \"value\"\r}\n",
            "ingestionTime": 1559763018353
        },
        {
            "timestamp": 1559763003218,
            "message": "END RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf\n",
            "ingestionTime": 1559763018353
        },
        {
            "timestamp": 1559763003218,
            "message": "REPORT RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tDuration: 26.73 ms\tBilled Duration: 27 ms \tMemory Size: 128 MB\tMax Memory Used: 75
 MB\t\n",
            "ingestionTime": 1559763018353
        }
    ],
    "nextForwardToken": "f/34783877304859518393868359594929986069206639495374241795",
    "nextBackwardToken": "b/34783877303811383369537420289090800615709599058929582080"
}
```

Menghapus log

Grup log tidak terhapus secara otomatis ketika Anda menghapus suatu fungsi. Untuk menghindari penyimpanan log secara tidak terbatas, hapus kelompok log, atau lakukan konfigurasi periode penyimpanan, yang setelahnya log akan dihapus secara otomatis.

Menginstrumentasi Node.js kode di AWS Lambda

Lambda terintegrasi dengan AWS X-Ray untuk membantu Anda melacak, men-debug, dan mengoptimalkan aplikasi Lambda. Anda dapat menggunakan X-Ray untuk melacak permintaan saat melintasi sumber daya dalam aplikasi Anda, yang mungkin termasuk fungsi Lambda dan layanan lainnya. AWS

Untuk mengirim data penelusuran X-Ray, Anda dapat menggunakan salah satu dari dua pustaka SDK:

- AWS Distro for OpenTelemetry (ADOT) — Distribusi SDK (OTel) yang aman, siap produksi, dan AWS didukung. OpenTelemetry
- AWS X-Ray SDK for Node.js— SDK untuk menghasilkan dan mengirim data jejak ke X-Ray.

Setiap SDK menawarkan cara untuk mengirim data telemetri Anda ke layanan. X-Ray Anda kemudian dapat menggunakan X-Ray untuk melihat, memfilter, dan mendapatkan wawasan tentang metrik kinerja aplikasi Anda untuk mengidentifikasi masalah dan peluang untuk pengoptimalan.

> **Important**
>
> The X-Ray dan Powertools untuk AWS Lambda SDK adalah bagian dari solusi instrumentasi terintegrasi ketat yang ditawarkan oleh. AWS Lapisan Lambda ADOT adalah bagian dari standar industri untuk melacak instrumentasi yang mengumpulkan lebih banyak data secara umum, tetapi mungkin tidak cocok untuk semua kasus penggunaan. Anda dapat menerapkan penelusuran ujung ke ujung dalam X-Ray menggunakan salah satu solusi. Untuk mempelajari lebih lanjut tentang memilih di antara keduanya, lihat Memilih antara AWS Distro untuk Telemetri Terbuka dan SDK. X-Ray

Bagian-bagian

- Menggunakan ADOT untuk instrumen fungsi Anda Node.js
- Menggunakan X-Ray SDK untuk menginstruksikan fungsi Anda Node.js
- Mengaktifkan penelusuran dengan konsol Lambda
- Mengaktifkan penelusuran dengan Lambda API
- Mengaktifkan penelusuran dengan CloudFormation
- Menafsirkan jejak X-Ray

- Menyimpan dependensi runtime dalam lapisan (SDK) X-Ray

Menggunakan ADOT untuk instrumen fungsi Anda Node.js

ADOT menyediakan lapisan Lambda yang dikelola sepenuhnya yang mengemas semua yang Anda butuhkan untuk mengumpulkan data telemetri menggunakan OTel SDK. Dengan menggunakan lapisan ini, Anda dapat menginstruksikan fungsi Lambda Anda tanpa harus memodifikasi kode fungsi apa pun. Anda juga dapat mengonfigurasi lapisan Anda untuk melakukan inisialisasi khusus OTel. Untuk informasi selengkapnya, lihat Konfigurasi khusus untuk Kolektor ADOT di Lambda dalam dokumentasi ADOT.

Untuk Node.js runtime, Anda dapat menambahkan layer Lambda AWS terkelola untuk ADOT Javascript untuk menginstruksikan fungsi Anda secara otomatis. Untuk petunjuk rinci tentang cara menambahkan layer ini, lihat AWS Distro untuk OpenTelemetry Lambda JavaScript Support untuk dalam dokumentasi ADOT.

Menggunakan X-Ray SDK untuk menginstruksikan fungsi Anda Node.js

Untuk merekam detail tentang panggilan yang dilakukan oleh Fungsi Peristiwa Lambda ke sumber daya lain dalam aplikasi, Anda juga dapat menggunakan. AWS X-Ray SDK for Node.js Untuk mendapatkan SDK, tambahkan paket `aws-xray-sdk-core` ke dependensi aplikasi Anda.

Example kosong- nodejs/package .json

```
{
  "name": "blank-nodejs",
  "version": "1.0.0",
  "private": true,
  "devDependencies": {
    "jest": "29.7.0"
  },
  "dependencies": {
    "@aws-sdk/client-lambda": "3.345.0",
    "aws-xray-sdk-core": "3.5.3"
  },
  "scripts": {
    "test": "jest"
  }
}
```

Untuk AWS menginstrumentasikan klien SDK di AWS SDK untuk JavaScript v3, bungkus instance klien dengan metode. `captureAWSv3Client`

Example kosong- nodejs/function /index.js — Menelusuri AWS Klien SDK

```
const AWSXRay = require('aws-xray-sdk-core');
const { LambdaClient, GetAccountSettingsCommand } = require('@aws-sdk/client-lambda');

// Create client outside of handler to reuse
const lambda = AWSXRay.captureAWSv3Client(new LambdaClient());

// Handler
exports.handler = async function(event, context) {
    event.Records.forEach(record => {
  ...
```

Runtime Lambda menetapkan beberapa variabel lingkungan untuk mengonfigurasi SDK. X-Ray Misalnya, Lambda disetel `AWS_XRAY_CONTEXT_MISSING` `LOG_ERROR` untuk menghindari kesalahan runtime dari SDK. X-Ray Untuk menetapkan strategi yang hilang dalam konteks khusus, timpa variabel lingkungan dalam konfigurasi fungsi Anda agar tidak memiliki nilai, lalu Anda dapat menetapkan konteks yang tidak memiliki strategi yang terprogram.

Example Kode inisialisasi contoh

```
const AWSXRay = require('aws-xray-sdk-core');

// Configure the context missing strategy to do nothing
AWSXRay.setContextMissingStrategy(() => {});
```

Untuk informasi selengkapnya, lihat the section called “Variabel lingkungan”.

Setelah Anda menambahkan dependensi yang benar dan membuat perubahan kode yang diperlukan, aktifkan penelusuran dalam konfigurasi fungsi Anda melalui konsol Lambda atau API.

Mengaktifkan penelusuran dengan konsol Lambda

Untuk mengaktifkan penelusuran aktif di Fungsi Peristiwa Lambda Anda dengan konsol, ikuti langkah-langkah berikut:

Untuk mengaktifkan penelusuran aktif

1. Buka halaman Fungsi Acara konsol Lambda.

2. Pilih Fungsi Acara.
3. Pilih Konfigurasi dan kemudian pilih Alat Pemantauan dan operasi.
4. Di bawah Alat pemantauan tambahan, pilih Edit.
5. Di bawah Sinyal CloudWatch Aplikasi dan AWS X-Ray, pilih Aktifkan untuk jejak layanan Lambda.
6. Pilih Simpan.

Mengaktifkan penelusuran dengan Lambda API

Konfigurasikan penelusuran pada fungsi Lambda Anda dengan AWS CLI AWS atau SDK, gunakan operasi API berikut:

- UpdateFunctionConfiguration
- GetFunctionConfiguration
- CreateFunction

Contoh AWS CLI perintah berikut memungkinkan penelusuran aktif pada fungsi bernama my-function.

```
aws lambda update-function-configuration --function-name my-function \
--tracing-config Mode=Active
```

Mode penelusuran adalah bagian dari konfigurasi khusus versi saat Anda mempublikasikan versi fungsi Anda. Anda tidak dapat mengubah mode pelacakan pada versi yang telah diterbitkan.

Mengaktifkan penelusuran dengan CloudFormation

Untuk mengaktifkan penelusuran pada `AWS::Lambda::Function` sumber daya dalam CloudFormation templat, gunakan `TracingConfig` properti.

Example fungsi-inline.yml. — Konfigurasi penelusuran

```
Resources:
  function:
    Type: AWS::Lambda::Function
    Properties:
      TracingConfig:
        Mode: Active
```

```
      ...
```

Untuk sumber `AWS::Serverless::Function` daya AWS Serverless Application Model (AWS SAM), gunakan `Tracing` properti.

Example template.yml — Konfigurasi penelusuran

```
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      Tracing: Active
      ...
```

Menafsirkan jejak X-Ray

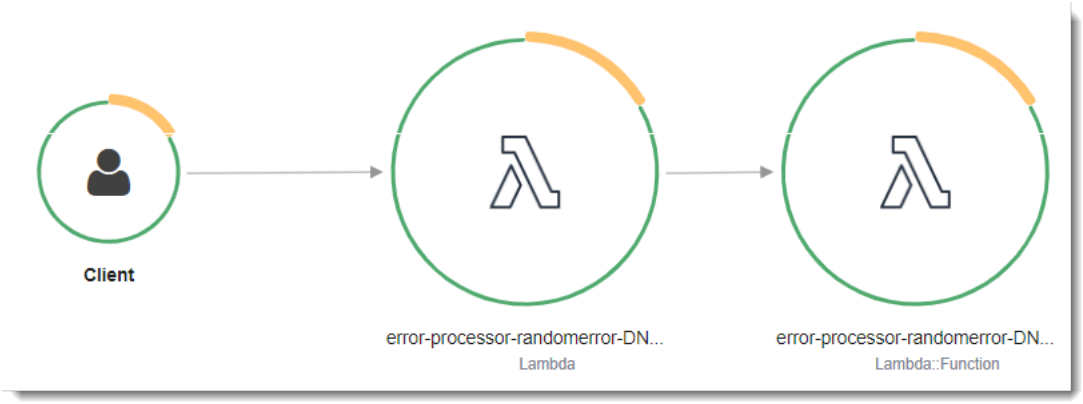
Fungsi Acara Anda memerlukan izin untuk mengunggah data jejak ke X-Ray. Saat Anda mengaktifkan penelusuran di konsol Lambda, Lambda menambahkan izin yang diperlukan ke peran eksekusi Fungsi Acara Anda. Atau, tambahkan kebijakan AWSXRayDaemonWriteAccess ke peran eksekusi.

Setelah mengonfigurasi penelusuran aktif, Anda dapat mengamati permintaan tertentu melalui aplikasi Anda. Grafik X-Ray layanan menunjukkan informasi tentang aplikasi Anda dan semua komponennya. Contoh berikut menunjukkan aplikasi dengan dua fungsi. Fungsi utama memproses kejadian dan terkadang mengembalikan kesalahan. Fungsi kedua di bagian atas memproses kesalahan yang muncul di grup log pertama dan menggunakan AWS SDK untuk memanggil X-Ray, Amazon Simple Storage Service (Amazon S3) Service, dan Amazon Logs. CloudWatch

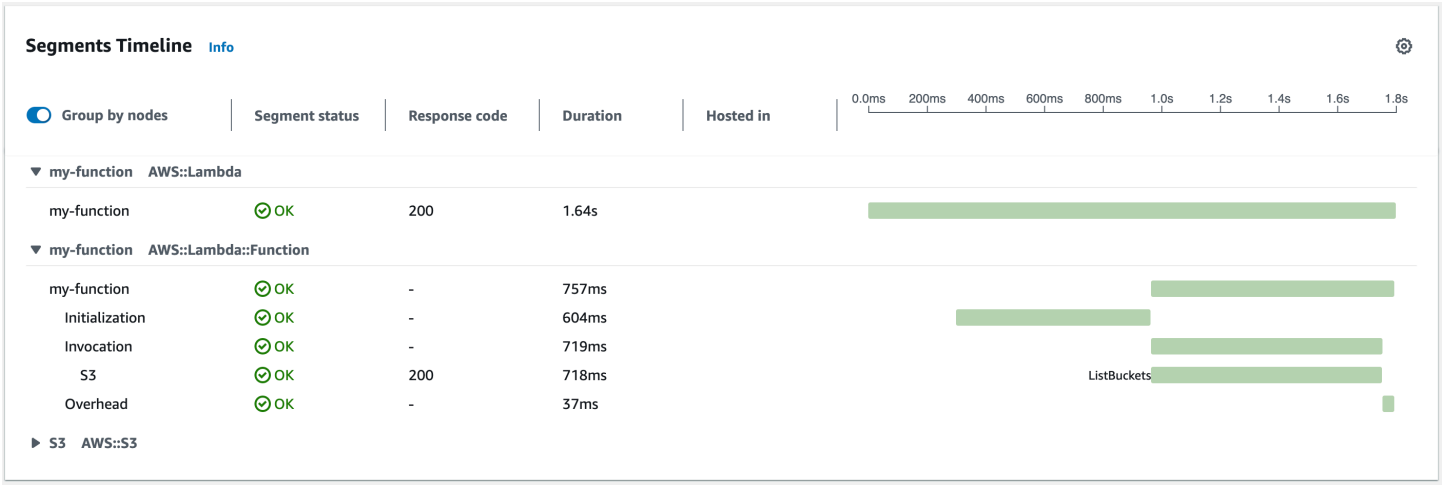


X-Ray tidak melacak semua permintaan ke aplikasi Anda. X-Ray menerapkan algoritma pengambilan sampel untuk memastikan bahwa penelusuran efisien, sambil tetap memberikan sampel yang representatif dari semua permintaan. Tingkat pengambilan sampel adalah 1 permintaan per detik dan 5 persen dari permintaan tambahan. Anda tidak dapat mengonfigurasi laju X-Ray pengambilan sampel untuk fungsi Anda.

Di X-Ray, jejak mencatat informasi tentang permintaan yang diproses oleh satu atau beberapa layanan. Lambda mencatat 2 segmen per jejak, yang menciptakan dua node pada grafik layanan. Gambar berikut menyoroti dua node ini:



Node pertama di sebelah kiri mewakili layanan Lambda, yang menerima permintaan pemanggilan. Node kedua mewakili fungsi Lambda spesifik Anda. Contoh berikut menunjukkan jejak dengan dua segmen ini. Keduanya bernama fungsi saya, tetapi yang satu memiliki asal `AWS::Lambda` dan yang lainnya memiliki asal usul. `AWS::Lambda::Function` Jika `AWS::Lambda` segmen menunjukkan kesalahan, layanan Lambda mengalami masalah. Jika `AWS::Lambda::Function` segmen menunjukkan kesalahan, fungsi Anda mengalami masalah.



Contoh ini memperluas `AWS::Lambda::Function` segmen untuk menunjukkan tiga subsegmennya.

> **Note**
>
> AWS saat ini menerapkan perubahan pada layanan Lambda. Karena perubahan ini, Anda mungkin melihat perbedaan kecil antara struktur dan konten pesan log sistem dan segmen jejak yang dipancarkan oleh Fungsi Peristiwa Lambda yang berbeda di Anda. Akun AWS Contoh jejak yang ditunjukkan di sini menggambarkan segmen fungsi gaya lama. Perbedaan antara segmen gaya lama dan baru dijelaskan dalam paragraf berikut.
> Perubahan ini akan diterapkan selama beberapa minggu mendatang, dan semua fungsi di semua Wilayah AWS kecuali China GovCloud dan wilayah akan bertransisi untuk menggunakan pesan log format baru dan segmen pelacakan.

Segmen fungsi gaya lama berisi subsegmen berikut:

- Inisialisasi – Mewakili waktu yang dihabiskan untuk memuat fungsi dan menjalankan kode inisialisasi. Subsegmen ini hanya muncul untuk peristiwa pertama yang diproses oleh setiap instance fungsi Anda.
- Doa - Merupakan waktu yang dihabiskan untuk menjalankan kode handler Anda.
- Overhead - Merupakan waktu yang dihabiskan runtime Lambda untuk mempersiapkan diri untuk menangani acara berikutnya.

Segmen fungsi gaya baru tidak berisi subsegmen. `Invocation` Sebagai gantinya, subsegmen pelanggan dilampirkan langsung ke segmen fungsi. Untuk informasi lebih lanjut tentang struktur segmen fungsi gaya lama dan baru, lihat. the section called “Memahami X-Ray jejak”

Anda juga dapat melakukan instrumentasi klien HTTP, merekam kueri SQL, dan membuat subsegmen khusus dengan anotasi dan metadata. Untuk informasi selengkapnya, lihat AWS X-Ray SDK for Node.jsdi Panduan AWS X-Ray Pengembang.

> **Harga**
>
> Anda dapat menggunakan X-Ray tracing gratis setiap bulan hingga batas tertentu sebagai bagian dari Tingkat AWS Gratis. Di luar ambang batas itu, X-Ray biaya untuk penyimpanan dan pengambilan jejak. Untuk informasi selengkapnya, lihat harga AWS X-Ray.

Menyimpan dependensi runtime dalam lapisan (SDK) X-Ray

Jika Anda menggunakan X-Ray SDK untuk menginstruksikan klien AWS SDK kode fungsi Anda, paket penerapan Anda bisa menjadi cukup besar. Untuk menghindari mengunggah dependensi runtime setiap kali Anda memperbarui kode fungsi, paketkan X-Ray SDK dalam lapisan Lambda.

Contoh berikut menunjukkan `AWS::Serverless::LayerVersion` sumber daya yang menyimpan file AWS X-Ray SDK for Node.js.

Example template.yml - Lapisan Dependensi

```
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: function/.
      Tracing: Active
      Layers:
        - !Ref libs
      ...
  libs:
    Type: AWS::Serverless::LayerVersion
    Properties:
      LayerName: blank-nodejs-lib
      Description: Dependencies for the blank sample app.
      ContentUri: lib/.
      CompatibleRuntimes:
        - nodejs24.x
```

Dengan konfigurasi ini, Anda memperbarui lapisan pustaka hanya jika Anda mengubah dependensi runtime Anda. Karena paket penerapan fungsi hanya berisi kode Anda, ini dapat membantu mengurangi waktu upload.

Membuat lapisan untuk dependensi memerlukan perubahan build untuk menghasilkan arsip lapisan sebelum penerapan. Untuk contoh pekerjaan, lihat contoh aplikasi blank-nodejs.

Membangun Fungsi Acara Lambda dengan TypeScript

Anda dapat menggunakan Node.js runtime untuk dijalankan TypeScript kode di AWS Lambda Fungsi Acara. Karena Node.js tidak berjalan TypeScript kode secara asli, Anda harus terlebih dahulu mentranspilasi TypeScript kode ke JavaScript. Kemudian, gunakan JavaScript file untuk menerapkan

kode fungsi Anda ke Lambda. Kode Anda berjalan di lingkungan yang mencakup AWS SDK untuk JavaScript, dengan kredensi dari sebuah AWS Identity and Access Management (IAM) peran yang Anda kelola. Untuk mempelajari lebih lanjut tentang versi SDK yang disertakan dengan Node.js runtime, lihat the section called “Runtime-included Versi SDK”.

Untuk informasi tentang penelusuran TypeScript fungsi dengan, lihat the section called “Pelacakan”.

Lambda mendukung yang berikut Node.js waktu proses.

Nama	Pengidentifikasi	Sistem operasi	Tanggal pengusangan	Blokir fungsi buat	Pembaruan fungsi blok
Node.js 26	`nodejs26.x`	Amazon Linux 2023	Tidak dijadwalkan	Tidak dijadwalkan	Tidak dijadwalkan
Node.js 24	`nodejs24.x`	Amazon Linux 2023	30 April 2028	1 Juni 2028	1 Juli 2028
Node.js 22	`nodejs22.x`	Amazon Linux 2023	30 April 2027	29 Juli 2027	31 Agu 2027

Topik

- Menyiapkan TypeScript lingkungan pengembangan
- Definisi tipe untuk Lambda
- Tentukan penangan Fungsi Peristiwa Lambda di TypeScript
- Terapkan TypeScript kode yang ditranspilasikan di Lambda dengan arsip file.zip
- Terapkan TypeScript kode yang ditranspilasikan di Lambda dengan gambar kontainer
- Menggunakan objek konteks Lambda untuk mengambil informasi Fungsi Acara TypeScript
- Log dan pantau TypeScript Fungsi Acara Lambda
- Menelusuri TypeScript kode di AWS Lambda

Menyiapkan TypeScript lingkungan pengembangan

Gunakan lingkungan pengembangan terintegrasi lokal (IDE) atau editor teks untuk menulis TypeScript kode fungsi. Anda tidak dapat membuat TypeScript kode pada konsol Lambda.

Anda dapat menggunakan salah satu esbuild atau milik Microsoft TypeScript kompilator (`tsc`) untuk mentranspilasi Anda TypeScript kode ke JavaScript. AWS Serverless Application Model (AWS SAM) dan AWS Cloud Development Kit (AWS CDK) keduanya menggunakan esbuild.

Saat menggunakan esbuild, pertimbangkan hal berikut:

- Ada beberapa TypeScript peringatan.
- Anda harus mengkonfigurasi TypeScript pengaturan transpilasi agar sesuai dengan Node.js runtime yang Anda rencanakan untuk digunakan. Untuk informasi lebih lanjut, lihat Target dalam dokumentasi esbuild. Sebagai contoh dari a tsconfig.json file yang menunjukkan cara menargetkan yang spesifik Node.js versi yang didukung oleh Lambda, lihat TypeScript GitHub repositori.
- esbuild tidak melakukan pemeriksaan tipe. Untuk memeriksa jenis, gunakan `tsc` kompiler. Jalankan `tsc -noEmit` atau tambahkan a `"noEmit"` parameter untuk Anda tsconfig.json file, seperti yang ditunjukkan pada contoh berikut. Ini mengonfigurasi `tsc` untuk tidak memancarkan JavaScript berkas. Setelah memeriksa jenis, gunakan esbuild untuk mengonversi TypeScript file ke JavaScript.

Example tsconfig.json

```
{
 "compilerOptions": {
   "target": "es2020",
   "strict": true,
   "preserveConstEnums": true,
   "noEmit": true,
   "sourceMap": false,
   "module":"commonjs",
   "moduleResolution":"node",
   "esModuleInterop": true,
   "skipLibCheck": true,
   "forceConsistentCasingInFileNames": true,
   "isolatedModules": true,
 },
 "exclude": ["node_modules", "**/*.test.ts"]
}
```

Definisi tipe untuk Lambda

@types/aws-Lambda paket menyediakan definisi tipe untuk Lambda Event Functions. Instal paket ini ketika fungsi Anda menggunakan salah satu dari berikut ini:

- Umum AWS sumber acara, seperti:
  - `APIGatewayProxyEvent`: Untuk Integrasi proxy Amazon API Gateway
  - `SNSEvent`: Untuk Pemberitahuan Layanan Pemberitahuan Sederhana Amazon
  - `SQSEvent`: Untuk Pesan Layanan Antrian Sederhana Amazon
  - `S3Event`: Untuk Peristiwa pemicu S3
  - `DynamoDBStreamEvent`: Untuk Aliran Amazon DynamoDB
- Lambda Konteks objek
- panggilan balik pola handler

Untuk menambahkan definisi tipe Lambda ke fungsi Anda, instal `@types/aws-lambda` sebagai ketergantungan pengembangan:

```
npm install -D @types/aws-lambda
```

Kemudian, impor tipe dari `aws-lambda`:

```
import { Context, S3Event, APIGatewayProxyEvent } from 'aws-lambda';

export const handler = async (event: S3Event, context: Context) => {
    // Function code
};
```

`import ... from 'aws-lambda'` pernyataan mengimpor definisi tipe. Itu tidak mengimpor `aws-lambda` paket npm, yang merupakan alat pihak ketiga yang tidak terkait. Untuk informasi lebih lanjut, lihat aws-lambda di DefinitelyTyped GitHub repositori.

> **Note**
>
> Anda tidak perlu @types/aws-Lambda saat menggunakan definisi tipe kustom Anda sendiri. Untuk contoh fungsi yang mendefinisikan jenisnya sendiri untuk objek peristiwa, lihat Contoh kode TypeScript Fungsi Acara Lambda.

Tentukan penangan Fungsi Peristiwa Lambda di TypeScript

Lambda Event Function handler adalah metode dalam kode fungsi Anda yang memproses peristiwa. Saat fungsi Anda diaktifkan, Lambda menjalankan metode handler. Fungsi Anda berjalan sampai handler mengembalikan respons, keluar, atau waktu habis.

Halaman ini menjelaskan cara bekerja dengan penangan Fungsi Peristiwa Lambda TypeScript, termasuk opsi untuk penyiapan proyek, konvensi penamaan, dan praktik terbaik. Halaman ini juga menyertakan contoh fungsi TypeScript Lambda yang mengambil informasi tentang pesanan, menghasilkan tanda terima file teks, dan menempatkan file ini di bucket Amazon Simple Storage Service (Amazon S3). Untuk informasi tentang cara menerapkan fungsi Anda setelah menulisnya, lihat the section called “Deploy arsip file .zip” atauthe section called “Deploy gambar kontainer”.

Topik

- Menyiapkan TypeScript proyek Anda
- Contoh kode TypeScript Fungsi Acara Lambda
- Modul CommonJS dan ES
- Node.js inisialisasi
- Konvensi penamaan handler
- Mendefinisikan dan mengakses objek peristiwa masukan
- Pola handler yang valid untuk fungsi TypeScript
- Menggunakan SDK untuk JavaScript v3 di handler Anda
- Mengakses variabel lingkungan
- Menggunakan status global
- Praktik terbaik kode untuk Fungsi TypeScript Acara Lambda

Menyiapkan TypeScript proyek Anda

Gunakan lingkungan pengembangan terintegrasi lokal (IDE) atau editor teks untuk menulis kode TypeScript fungsi Anda. Anda tidak dapat membuat TypeScript kode di konsol Lambda.

Ada beberapa cara untuk menginisialisasi proyek TypeScript Lambda. Misalnya, Anda dapat membuat proyek menggunakan`npm`, membuat AWS SAM aplikasi, atau membuat AWS CDK aplikasi. Untuk membuat proyek menggunakan`npm`:

```
npm init
```

Kode fungsi Anda berada dalam `.ts` file, yang Anda transpilasikan menjadi JavaScript file pada waktu pembuatan. Anda dapat menggunakan esbuild atau TypeScript compiler (`tsc`) Microsoft untuk mentranspile kode Anda TypeScript ke dalam. JavaScript Untuk menggunakan esbuild, tambahkan sebagai dependensi pengembangan:

```
npm install -D esbuild
```

Proyek fungsi TypeScript Lambda yang khas mengikuti struktur umum ini:

```
/project-root
  ### index.ts - Contains main handler
  ### dist/ - Contains compiled JavaScript
  ### package.json - Project metadata and dependencies
  ### package-lock.json - Dependency lock file
  ### tsconfig.json - TypeScript configuration
  ### node_modules/ - Installed dependencies
```

Contoh kode TypeScript Fungsi Acara Lambda

Contoh kode fungsi Lambda berikut mengambil informasi tentang pesanan, menghasilkan tanda terima file teks, dan menempatkan file ini di bucket Amazon S3. Contoh ini mendefinisikan jenis acara kustom (`OrderEvent`). Untuk mempelajari cara mengimpor definisi tipe untuk sumber AWS peristiwa, lihat Definisi jenis untuk Lambda.

> **Note**
>
> Contoh ini menggunakan penangan modul ES. Lambda mendukung modul ES dan penangan CommonJS. Untuk informasi selengkapnya, lihat Modul CommonJS dan ES.

Example index.ts Lambda Fungsi Acara

```
import { S3Client, PutObjectCommand } from '@aws-sdk/client-s3';

// Initialize the S3 client outside the handler for reuse
const s3Client = new S3Client();

// Define the shape of the input event
type OrderEvent = {
    order_id: string;
    amount: number;
```

```
    item: string;
}

/**
 * Lambda handler for processing orders and storing receipts in S3.
 */
export const handler = async (event: OrderEvent): Promise<string> => {
    try {
        // Access environment variables
        const bucketName = process.env.RECEIPT_BUCKET;
        if (!bucketName) {
            throw new Error('RECEIPT_BUCKET environment variable is not set');
        }

        // Create the receipt content and key destination
        const receiptContent = `OrderID: ${event.order_id}\nAmount: $
${event.amount.toFixed(2)}\nItem: ${event.item}`;
        const key = `receipts/${event.order_id}.txt`;

        // Upload the receipt to S3
        await uploadReceiptToS3(bucketName, key, receiptContent);

        console.log(`Successfully processed order ${event.order_id} and stored receipt
 in S3 bucket ${bucketName}`);
        return 'Success';
    } catch (error) {
        console.error(`Failed to process order: ${error instanceof Error ?
 error.message : 'Unknown error'}`);
        throw error;
    }
};

/**
 * Helper function to upload receipt to S3
 */
async function uploadReceiptToS3(bucketName: string, key: string, receiptContent:
 string): Promise<void> {
    try {
        const command = new PutObjectCommand({
            Bucket: bucketName,
            Key: key,
            Body: receiptContent
        });
```

```
        await s3Client.send(command);
    } catch (error) {
        throw new Error(`Failed to upload receipt to S3: ${error instanceof Error ?
 error.message : 'Unknown error'}`);
    }
}
```

`index.ts`File ini berisi bagian kode berikut:

- `import`block: Gunakan blok ini untuk menyertakan pustaka yang dibutuhkan fungsi Lambda Anda, AWS seperti klien SDK.
- `const s3Client`deklarasi: Ini menginisialisasi klien Amazon S3 di luar fungsi handler. Hal ini menyebabkan Lambda menjalankan kode ini selama fase inisialisasi, dan klien dipertahankan untuk digunakan kembali di beberapa pemanggilan.
- `type OrderEvent`: Mendefinisikan struktur peristiwa input yang diharapkan.
- `export const handler`: Ini adalah fungsi handler utama yang dipanggil Lambda. Saat menerapkan fungsi Anda, tentukan properti `index.handler` Handler. Nilai `Handler` properti adalah nama file dan nama metode handler yang diekspor, dipisahkan oleh sebuah titik.
- `uploadReceiptToS3`fungsi: Ini adalah fungsi pembantu yang direferensikan oleh fungsi handler utama.

Agar fungsi ini berfungsi dengan baik, peran pelaksanaannya harus memungkinkan `s3:PutObject` tindakan. Juga, pastikan bahwa Anda mendefinisikan variabel `RECEIPT_BUCKET` lingkungan. Setelah pemanggilan berhasil, bucket Amazon S3 harus berisi file tanda terima.

Modul CommonJS dan ES

Node.js mendukung dua sistem modul: modul CommonJS dan ECMAScript (modul ES). Lambda merekomendasikan penggunaan modul ES karena mendukung await tingkat atas, yang memungkinkan tugas asinkron diselesaikan selama inisialisasi lingkungan eksekusi.

Node.js memperlakukan file dengan ekstensi nama `.cjs` file sebagai modul CommonJS sementara `.mjs` ekstensi menunjukkan modul ES. Secara default, Node.js memperlakukan file dengan ekstensi nama `.js` file sebagai modul CommonJS. Anda dapat mengonfigurasi Node.js untuk memperlakukan `.js` file sebagai modul ES dengan menentukan `type` as `module` pada `package.json` file fungsi. Anda dapat mengonfigurasi Node.js di Lambda untuk mendeteksi secara otomatis apakah `.js` file harus diperlakukan sebagai CommonJS atau sebagai modul ES dengan menambahkan –

`experimental-detect-module` flag ke variabel lingkungan. `NODE_OPTIONS` Untuk informasi selengkapnya, lihat Node.js Fitur eksperimental.

Contoh berikut menunjukkan penangan fungsi yang ditulis menggunakan modul ES dan modul CommonJS. Contoh yang tersisa di halaman ini semuanya menggunakan modul ES.

Node.js inisialisasi

Node.js menggunakan I/O model non-pemblokiran yang mendukung operasi asinkron yang efisien menggunakan loop peristiwa. Misalnya, jika Node.js melakukan panggilan jaringan, fungsi terus memproses operasi lain tanpa memblokir respons jaringan. Ketika respon jaringan diterima, itu ditempatkan ke antrian callback. Tugas dari antrian diproses ketika tugas saat ini selesai.

Lambda merekomendasikan penggunaan await tingkat atas sehingga tugas asinkron yang dimulai selama inisialisasi lingkungan eksekusi diselesaikan selama inisialisasi. Tugas asinkron yang tidak selesai selama inisialisasi biasanya berjalan selama pemanggilan fungsi pertama. Ini dapat menyebabkan perilaku atau kesalahan yang tidak terduga.

Misalnya, inisialisasi fungsi Anda mungkin membuat panggilan jaringan untuk mengambil parameter dari AWS Parameter Store. Jika tugas ini tidak selesai selama inisialisasi, nilainya mungkin null selama pemanggilan. Mungkin juga ada penundaan antara inisialisasi dan pemanggilan yang dapat memicu kesalahan dalam operasi yang sensitif terhadap waktu. Secara khusus, panggilan AWS layanan dapat mengandalkan tanda tangan permintaan yang sensitif terhadap waktu, yang mengakibatkan kegagalan panggilan layanan jika panggilan tidak selesai selama fase inisialisasi.

Menyelesaikan tugas selama inisialisasi biasanya meningkatkan kinerja cold-start, dan pertama-tama memanggil kinerja saat menggunakan Provisioned Concurrency. Untuk informasi lebih lanjut, lihat posting blog kami Menggunakan modul Node.js ES dan menunggu tingkat atas. AWS Lambda

Konvensi penamaan handler

Saat Anda mengonfigurasi fungsi, nilai pengaturan Handler adalah nama file dan nama metode handler yang diekspor, dipisahkan oleh titik. Default untuk fungsi yang dibuat di konsol dan untuk contoh dalam panduan ini adalah`index.handler`. Ini menunjukkan `handler` metode yang diekspor dari `index.mjs` file `index.js` atau.

Jika Anda membuat fungsi di konsol menggunakan nama file atau nama pengendali fungsi yang berbeda, Anda harus mengedit nama handler default.

Untuk mengubah nama fungsi handler (konsol)

1. Buka halaman Fungsi di konsol Lambda dan pilih fungsi Anda.
2. Pilih tab Kode.
3. Gulir ke bawah ke panel pengaturan Runtime dan pilih Edit.
4. Di Handler, masukkan nama baru untuk handler fungsi Anda.
5. Pilih Simpan.

Mendefinisikan dan mengakses objek peristiwa masukan

JSON adalah format input yang paling umum dan standar untuk Lambda Event Functions. Dalam contoh ini, fungsi mengharapkan input yang mirip dengan yang berikut:

```
{
    "order_id": "12345",
    "amount": 199.99,
    "item": "Wireless Headphones"
}
```

Saat bekerja dengan fungsi Lambda di TypeScript, Anda dapat menentukan bentuk peristiwa input menggunakan tipe atau antarmuka. Dalam contoh ini, kita mendefinisikan struktur acara menggunakan tipe:

```
type OrderEvent = {
    order_id: string;
    amount: number;
    item: string;
}
```

Setelah Anda menentukan jenis atau antarmuka, gunakan itu dalam tanda tangan handler Anda untuk memastikan keamanan tipe:

```
export const handler = async (event: OrderEvent): Promise<string> => {
```

Selama kompilasi, TypeScript memvalidasi bahwa objek acara berisi bidang wajib dengan tipe yang benar. Misalnya, TypeScript kompiler melaporkan kesalahan jika Anda mencoba menggunakan `event.order_id` sebagai angka atau `event.amount` sebagai string.

Pola handler yang valid untuk fungsi TypeScript

Sebaiknya gunakan async/awaituntuk mendeklarasikan fungsi handler alih-alih menggunakan callback. Async/await adalah cara ringkas dan mudah dibaca untuk menulis kode asinkron, tanpa perlu panggilan balik bersarang atau janji rantai. Dengan async/await, Anda dapat menulis kode yang berbunyi seperti kode sinkron, sambil tetap asinkron dan tidak memblokir.

Contoh di bagian ini menggunakan `S3Event` tipe. Namun, Anda dapat menggunakan jenis AWS acara lain dalam paket @ types/aws -lambda, atau menentukan jenis acara Anda sendiri. Untuk menggunakan tipe dari@types/aws-lambda:

1. Tambahkan @types/aws-lambda paket sebagai dependensi pengembangan:

   ```
   npm install -D @types/aws-lambda
   ```

2. Impor jenis yang Anda butuhkan, seperti`Context`,`S3Event`, atau`Callback`.

penangan fungsi async (disarankan)

`async`Kata kunci menandai fungsi sebagai asinkron, dan `await` kata kunci menjeda eksekusi fungsi sampai a diselesaikan. `Promise` Handler menerima argumen berikut:

- `event`: Berisi data input yang diteruskan ke fungsi Anda.
- `context`: Berisi informasi tentang pemanggilan, fungsi, dan lingkungan eksekusi. Untuk informasi selengkapnya, lihat Menggunakan objek konteks Lambda untuk mengambil informasi Fungsi Acara TypeScript.

Berikut adalah tanda tangan yang valid untuk pola tersebut async/await :

```
export const handler = async (event: S3Event): Promise<void> => { };
```

```
export const handler = async (event: S3Event, context: Context): Promise<void> => { };
```

> **Note**
>
> Saat memproses array item secara asinkron, pastikan untuk menggunakan await with `Promise.all` untuk memastikan semua operasi selesai. Metode seperti `forEach`

jangan menunggu panggilan balik asinkron selesai. Untuk informasi selengkapnya, lihat Array.prototype.forEach() di dokumentasi Mozilla.

Penangan fungsi sinkron

Jika fungsi Anda tidak melakukan tugas asinkron, Anda dapat menggunakan penangan fungsi sinkron, menggunakan salah satu tanda tangan fungsi berikut:

```
export const handler = (event: S3Event): void => { };
```

```
export const handler = (event: S3Event, context: Context): void => { };
```

Respon penangan fungsi streaming

Lambda mendukung streaming respons dengan. Node.js Penangan fungsi streaming respons menggunakan dekorator AWSLAMBDA.streamifyResponse () dan mengambil 3 parameter: event, responseStream, dan konteks. Tanda tangan fungsi adalah:

```
export const handler = awslambda.streamifyResponse(async (event: APIGatewayProxyEvent,
 responseStream: NodeJS.WritableStream, context: Context) => { });
```

Untuk informasi selengkapnya, lihat Streaming respons untuk fungsi Lambda.

Callback-based penangan fungsi

> **Note**
>
> Callback-based penangan fungsi hanya didukung hingga Node.js 22. Mulai dari Node.js 24, tugas asinkron harus diimplementasikan menggunakan penangan fungsi asinkron.

Callback-based penangan fungsi dapat menggunakan argumen event, context, dan callback. Argumen callback mengharapkan `Error` dan respons, yang harus. JSON-serializable

Berikut adalah tanda tangan yang valid untuk pola handler callback:

```
export const handler = (event: S3Event, context: Context, callback: Callback<void>):
 void => { };
```

Fungsi terus mengeksekusi sampai loop peristiwa kosong atau fungsi habis waktu. Respons tidak dikirimkan ke invoker hingga semua tugas loop peristiwa selesai. Jika waktu fungsi habis, kesalahan akan dikembalikan. Anda dapat mengonfigurasi runtime untuk mengirim respons segera dengan menyetel WaitsForEmptyEventLoopcontext.callback ke false.

Example TypeScript fungsi dengan callback

Contoh berikut menggunakan`APIGatewayProxyCallback`, yang merupakan jenis callback khusus khusus untuk integrasi API Gateway. Sebagian besar sumber AWS acara menggunakan `Callback` tipe generik yang ditunjukkan pada tanda tangan di atas.

```
import { Context, APIGatewayProxyCallback, APIGatewayEvent } from 'aws-lambda';

export const lambdaHandler = (event: APIGatewayEvent, context: Context, callback:
 APIGatewayProxyCallback): void => {
    console.log(`Event: ${JSON.stringify(event, null, 2)}`);
    console.log(`Context: ${JSON.stringify(context, null, 2)}`);
    callback(null, {
        statusCode: 200,
        body: JSON.stringify({
            message: 'hello world',
        }),
    });
};
```

Menggunakan SDK untuk JavaScript v3 di handler Anda

Seringkali, Anda akan menggunakan fungsi Lambda untuk berinteraksi dengan atau membuat pembaruan ke sumber daya lain AWS . Cara termudah untuk berinteraksi dengan sumber daya ini adalah dengan menggunakan file AWS SDK untuk JavaScript. Semua Node.js runtime Lambda yang didukung menyertakan SDK untuk versi 3. JavaScript Namun, kami sangat menyarankan agar Anda menyertakan klien AWS SDK yang Anda butuhkan dalam paket penerapan Anda. Ini memaksimalkan kompatibilitas mundur selama pembaruan runtime Lambda di masa mendatang.

Untuk menambahkan dependensi SDK ke fungsi Anda, gunakan `npm install` perintah untuk klien SDK tertentu yang Anda butuhkan. Dalam kode contoh, kami menggunakan klien Amazon S3. Tambahkan dependensi ini dengan menjalankan perintah berikut di direktori yang berisi `package.json` file Anda:

```
npm install @aws-sdk/client-s3
```

Dalam kode fungsi, impor klien dan perintah yang Anda butuhkan, seperti yang ditunjukkan oleh fungsi contoh:

```
import { S3Client, PutObjectCommand } from '@aws-sdk/client-s3';
```

Kemudian, inisialisasi klien Amazon S3:

```
const s3Client = new S3Client();
```

Dalam contoh ini, kami menginisialisasi klien Amazon S3 kami di luar fungsi pengendali utama untuk menghindari keharusan menginisialisasi setiap kali kami menjalankan fungsi kami. Setelah menginisialisasi klien SDK, Anda dapat menggunakannya untuk melakukan panggilan API untuk layanan tersebut AWS . Kode contoh memanggil tindakan PutObjectAPI Amazon S3 sebagai berikut:

```
const command = new PutObjectCommand({
    Bucket: bucketName,
    Key: key,
    Body: receiptContent
});
```

Mengakses variabel lingkungan

Dalam kode handler Anda, Anda dapat mereferensikan variabel lingkungan apa pun dengan menggunakan`process.env`. Dalam contoh ini, kami mereferensikan variabel `RECEIPT_BUCKET` lingkungan yang ditentukan menggunakan baris kode berikut:

```
// Access environment variables
const bucketName = process.env.RECEIPT_BUCKET;
if (!bucketName) {
    throw new Error('RECEIPT_BUCKET environment variable is not set');
}
```

Menggunakan status global

Lambda menjalankan kode statis Anda selama fase inisialisasi sebelum menjalankan fungsi Anda untuk pertama kalinya. Sumber daya yang dibuat selama inisialisasi tetap berada di memori di antara pemanggilan, sehingga Anda dapat menghindari keharusan membuatnya setiap kali Anda menjalankan fungsi Anda.

Dalam kode contoh, kode inisialisasi klien S3 berada di luar handler. Runtime menginisialisasi klien sebelum fungsi menangani peristiwa pertamanya, dan klien tetap tersedia untuk digunakan kembali di semua pemanggilan.

Praktik terbaik kode untuk Fungsi TypeScript Acara Lambda

Ikuti panduan ini saat membangun fungsi Lambda:

- Pisahkan handler Lambda dari logika inti Anda. Ini memungkinkan Anda untuk membuat fungsi yang lebih dapat teruji.
- Kontrol dependensi dalam paket penerapan fungsi Anda. Lingkungan AWS Lambda eksekusi berisi sejumlah pustaka. Untuk runtime Node.js dan Python, ini termasuk SDK. AWS Untuk mengaktifkan serangkaian fitur dan pembaruan keamanan terbaru, Lambda akan memperbarui pustaka ini secara berkala. Pembaruan ini dapat memberikan perubahan kecil pada perilaku fungsi Lambda Anda. Untuk memiliki kendali penuh atas dependensi yang digunakan fungsi Anda, kemas semua dependensi Anda dengan paket deployment Anda.
- Minimalkan kompleksitas dependensi Anda. Utamakan memilih kerangka kerja lebih sederhana yang cepat dimuat dalam memulai lingkungan eksekusi.
- Minimalkan ukuran paket penerapan Anda sesuai kebutuhan runtime-nya. Ini akan mengurangi jumlah waktu yang dibutuhkan untuk mengunduh dan membongkar paket deployment Anda sebelum invokasi.

Manfaatkan penggunaan kembali lingkungan eksekusi untuk meningkatkan kinerja fungsi Anda. Inisialisasi klien SDK dan koneksi basis data di luar fungsi handler, dan lakukan caching aset statis secara lokal di direktori `/tmp`. Invokasi selanjutnya yang diproses oleh instans yang sama dari fungsi Anda dapat menggunakan kembali sumber daya ini. Ini menghemat biaya dengan mengurangi waktu pengoperasian fungsi.

Untuk menghindari potensi kebocoran data di seluruh invokasi, jangan menggunakan lingkungan eksekusi untuk menyimpan data pengguna, peristiwa, atau informasi lainnya implikasi keamanan. Jika fungsi Anda bergantung pada status yang dapat disenyapkan yang tidak dapat disimpan dalam memori di dalam handler, pertimbangkan untuk membuat fungsi terpisah atau versi terpisah dari fungsi untuk setiap pengguna.

Gunakan arahan keep-alive untuk mempertahankan koneksi yang persisten. Lambda membersihkan koneksi idle dari waktu ke waktu. Mencoba menggunakan ulang koneksi idle saat mengidentifikasi suatu fungsi akan menyebabkan kesalahan koneksi. Untuk mempertahankan koneksi yang

persisten, gunakan arahan tetap aktif yang berkaitan dengan runtime Anda. Sebagai contoh, lihat Menggunakan Kembali Koneksi dengan Keep-Alive in Node.js.

Gunakan variabel lingkungan untuk meneruskan parameter operasional ke fungsi Anda. Misalnya, jika Anda ingin menulis ke bucket Amazon S3 alih-alih melakukan hard-coding nama bucket yang Anda tulis, konfigurasikan nama bucket sebagai variabel lingkungan.

Hindari menggunakan pemanggilan rekursif dalam fungsi Lambda Anda, di mana fungsi memanggil dirinya sendiri atau memulai proses yang dapat memanggil fungsi lagi. Hal ini dapat menyebabkan volume invokasi fungsi yang tidak diinginkan dan peningkatan biaya. Jika Anda melihat volume pemanggilan yang tidak diinginkan, setel konkurensi fungsi cadangan untuk `0` segera membatasi semua pemanggilan ke fungsi, saat Anda memperbarui kode.

Jangan gunakan API non-publik yang tidak terdokumentasi dalam kode fungsi Lambda Anda. Untuk runtime AWS Lambda terkelola, Lambda secara berkala menerapkan pembaruan keamanan dan fungsional ke API internal Lambda. Pembaruan API internal ini mungkin tidak kompatibel ke belakang, yang menyebabkan konsekuensi yang tidak diinginkan seperti kegagalan pemanggilan jika fungsi Anda memiliki ketergantungan pada API non-publik ini. Lihat referensi API untuk daftar API yang tersedia untuk umum.

Tulis kode idempoten. Menulis kode idempoten untuk fungsi Anda memastikan bahwa peristiwa duplikat ditangani dengan cara yang sama. Kode Anda harus memvalidasi peristiwa dengan benar dan menangani peristiwa duplikat dengan anggun. Untuk informasi selengkapnya, lihat Bagaimana cara membuat fungsi Lambda saya idempoten? .

Terapkan TypeScript kode yang ditranspilasikan di Lambda dengan arsip file.zip

Sebelum Anda dapat menerapkan TypeScript kode ke AWS Lambda, Anda perlu mentranspilasinya menjadi. JavaScript Halaman ini menjelaskan tiga cara untuk membangun dan menyebarkan TypeScript kode ke Lambda dengan arsip file.zip:

- Menggunakan AWS Serverless Application Model (AWS SAM)
- Menggunakan AWS Cloud Development Kit (AWS CDK)
- Menggunakan AWS Command Line Interface (AWS CLI) dan esbuild

AWS SAM dan AWS CDK menyederhanakan fungsi membangun dan menyebarkan TypeScript . Spesifikasi AWS SAM template menyediakan sintaks sederhana dan bersih untuk menggambarkan fungsi Lambda, API, izin, konfigurasi, dan peristiwa yang membentuk aplikasi tanpa server Anda. Dengan ini AWS CDK, Anda dapat membangun aplikasi yang andal, terukur, dan hemat biaya di cloud dengan kekuatan ekspresif yang cukup besar dari bahasa pemrograman. AWS CDK Ini ditujukan untuk AWS pengguna yang cukup hingga sangat berpengalaman. Baik esbuild AWS CDK dan AWS SAM use untuk mentranspile TypeScript kode menjadi. JavaScript

Penggunaan AWS SAM untuk menyebarkan TypeScript kode ke Lambda

Ikuti langkah-langkah di bawah ini untuk mengunduh, membangun, dan menyebarkan contoh aplikasi Hello World menggunakan TypeScript aplikasi Hello World. AWS SAM Aplikasi ini mengimplementasikan backend API dasar. Ini terdiri dari titik akhir Amazon API Gateway dan Fungsi Peristiwa Lambda. Ketika Anda mengirim permintaan GET ke titik akhir API Gateway, fungsi Lambda dipanggil. Fungsi mengembalikan `hello world` pesan.

> **Note**
>
> AWS SAM menggunakan esbuild untuk membuat fungsi Node.js Lambda TypeScript dari kode. dukungan esbuild saat ini dalam pratinjau publik. Selama pratinjau publik, dukungan esbuild mungkin mengalami perubahan yang tidak kompatibel ke belakang.

Prasyarat

Untuk menyelesaikan langkah-langkah di bagian ini, Anda harus memiliki hal-hal berikut:

- AWS CLI versi 2

- AWS SAM CLI versi 1.75 atau yang lebih baru
- Node.js

Menyebarkan sampel AWS SAM application

1. Inisialisasi aplikasi menggunakan TypeScript template Hello World.

```
sam init --app-template hello-world-typescript --name sam-app --package-type Zip --runtime nodejs24.x
```

2. (Opsional) Aplikasi sampel mencakup konfigurasi untuk alat yang umum digunakan, seperti ESLLint untuk linting kode dan Jest untuk pengujian unit. Untuk menjalankan perintah lint dan test:

```
cd sam-app/hello-world
npm install
npm run lint
npm run test
```

3. Bangun aplikasi.

```
cd sam-app
sam build
```

4. Terapkan aplikasi.

```
sam deploy --guided
```

5. Ikuti petunjuk di layar. Untuk menerima opsi default yang disediakan dalam pengalaman interaktif, respons dengan `Enter`.
6. Output menunjukkan titik akhir untuk REST API. Buka titik akhir di browser untuk menguji fungsinya. Anda akan melihat tanggapan ini:

```
{"message":"hello world"}
```

7. Ini adalah titik akhir API publik yang dapat diakses melalui internet. Kami menyarankan Anda menghapus titik akhir setelah pengujian.

```
sam delete
```

Menggunakan AWS CDK untuk menyebarkan TypeScript kode ke Lambda

Ikuti langkah-langkah di bawah ini untuk membangun dan menerapkan TypeScript aplikasi sampel menggunakan aplikasi. AWS CDK Aplikasi ini mengimplementasikan backend API dasar. Ini terdiri dari titik akhir API Gateway dan fungsi Lambda. Ketika Anda mengirim permintaan GET ke titik akhir API Gateway, fungsi Lambda dipanggil. Fungsi mengembalikan `hello world` pesan.

Prasyarat

Untuk menyelesaikan langkah-langkah di bagian ini, Anda harus memiliki hal-hal berikut:

- AWS CLI versi 2
- AWS CDK versi 2
- Node.js
- Baik Docker atau esbuild

Menyebarkan sampel AWS CDK application

1. Buat direktori proyek untuk aplikasi baru Anda.

   ```
   mkdir hello-world
   cd hello-world
   ```

2. Inisialisasi aplikasi.

   ```
   cdk init app --language typescript
   ```

3. Tambahkan paket @ types/aws -lambda sebagai dependensi pengembangan. Paket ini berisi definisi tipe untuk Lambda.

   ```
   npm install -D @types/aws-lambda
   ```

4. Buka direktori lib. Anda akan melihat file bernama hello-world-stack.ts. Buat dua file baru di direktori ini: hello-world.function.ts dan hello-world.ts.
5. Buka hello-world.function.ts dan tambahkan kode berikut ke file. Ini adalah kode untuk fungsi Lambda.

> **Note**
>
> `import`Pernyataan mengimpor definisi tipe dari @ types/aws -lambda. Itu tidak mengimpor paket `aws-lambda` NPM, yang merupakan alat pihak ketiga yang tidak terkait. Untuk informasi selengkapnya, lihat aws-lambda di repositori. DefinitelyTyped GitHub

```
import { Context, APIGatewayProxyResult, APIGatewayEvent } from 'aws-lambda';

export const handler = async (event: APIGatewayEvent, context: Context):
 Promise<APIGatewayProxyResult> => {
    console.log(`Event: ${JSON.stringify(event, null, 2)}`);
    console.log(`Context: ${JSON.stringify(context, null, 2)}`);
    return {
        statusCode: 200,
        body: JSON.stringify({
            message: 'hello world',
        }),
    };
};
```

6. Buka hello-world.ts dan tambahkan kode berikut ke file. Ini berisi NodejsFunction konstruksi, yang menciptakan fungsi Lambda, dan konstruksi, LambdaRestApi yang membuat REST API.

```
import { Construct } from 'constructs';
import { NodejsFunction } from 'aws-cdk-lib/aws-lambda-nodejs';
import { LambdaRestApi } from 'aws-cdk-lib/aws-apigateway';

export class HelloWorld extends Construct {
  constructor(scope: Construct, id: string) {
    super(scope, id);
    const helloFunction = new NodejsFunction(this, 'function');
    new LambdaRestApi(this, 'apigw', {
      handler: helloFunction,
    });
  }
}
```

`NodejsFunction`Konstruk mengasumsikan hal berikut secara default:

- Handler fungsi Anda dipanggil`handler`.
- File.ts yang berisi kode fungsi (hello-world.function.ts) berada di direktori yang sama dengan file.ts yang berisi konstruksi (hello-world.ts). Konstruk menggunakan ID konstruksi (“hello-world”) dan nama file penangan Lambda (“fungsi”) untuk menemukan kode fungsi. Misalnya, jika kode fungsi Anda ada dalam file bernama hello-world.my-function.ts, file hello-world.ts harus mereferensikan kode fungsi seperti ini:

```
const helloFunction = new NodejsFunction(this, 'my-function');
```

Anda dapat mengubah perilaku ini dan mengonfigurasi parameter esbuild lainnya. Untuk informasi selengkapnya, lihat Mengonfigurasi esbuild di referensi AWS CDK API.

7. Buka hello-world-stack.ts. Ini adalah kode yang mendefinisikan AWS CDK tumpukan Anda. Ganti kode dengan yang berikut ini:

```
import { Stack, StackProps } from 'aws-cdk-lib';
import { Construct } from 'constructs';
import { HelloWorld } from './hello-world';

export class HelloWorldStack extends Stack {
  constructor(scope: Construct, id: string, props?: StackProps) {
    super(scope, id, props);
    new HelloWorld(this, 'hello-world');
  }
}
```

8. dari `hello-world` direktori yang berisi `cdk.json` file Anda, terapkan aplikasi Anda.

```
cdk deploy
```

9. AWS CDK Membangun dan mengemas fungsi Lambda menggunakan esbuild, lalu menerapkan fungsi tersebut ke runtime Lambda. Output menunjukkan titik akhir untuk REST API. Buka titik akhir di browser untuk menguji fungsinya. Anda akan melihat tanggapan ini:

```
{"message":"hello world"}
```

Ini adalah titik akhir API publik yang dapat diakses melalui internet. Kami menyarankan Anda menghapus titik akhir setelah pengujian.

Menggunakan AWS CLI dan esbuild untuk menyebarkan TypeScript kode ke Lambda

Contoh berikut menunjukkan cara mentranspile dan menyebarkan kode TypeScript ke Lambda menggunakan esbuild dan. esbuild menghasilkan satu file dengan semua AWS CLI dependensi. JavaScript Ini adalah satu-satunya file yang perlu Anda tambahkan ke arsip.zip.

Prasyarat

Untuk menyelesaikan langkah-langkah di bagian ini, Anda harus memiliki hal-hal berikut:

- AWS CLI versi 2
- Node.js
- Peran eksekusi untuk fungsi Lambda
- Untuk pengguna Windows, utilitas file zip seperti 7zip.

Menerapkan fungsi sampel

1. Pada mesin lokal Anda, buat direktori proyek untuk fungsi baru Anda.
2. Buat Node.js proyek baru dengan npm atau manajer paket pilihan Anda.

   ```
   npm init
   ```

3. Tambahkan paket @ types/aws -lambda dan esbuild sebagai dependensi pengembangan. `@types/aws-lambda`Paket berisi definisi tipe untuk Lambda.

   ```
   npm install -D @types/aws-lambda esbuild
   ```

4. Buat file baru bernama index.ts. Tambahkan kode berikut ke file baru. Ini adalah kode untuk fungsi Lambda. Fungsi mengembalikan `hello world` pesan. Fungsi ini tidak membuat sumber daya API Gateway apa pun.

   > **Note**
   >
   > `import`Pernyataan mengimpor definisi tipe dari @ types/aws -lambda. Itu tidak mengimpor paket `aws-lambda` NPM, yang merupakan alat pihak ketiga yang tidak terkait. Untuk informasi selengkapnya, lihat aws-lambda di repositori. DefinitelyTyped GitHub

```
import { Context, APIGatewayProxyResult, APIGatewayEvent } from 'aws-lambda';

export const handler = async (event: APIGatewayEvent, context: Context):
 Promise<APIGatewayProxyResult> => {
  console.log(`Event: ${JSON.stringify(event, null, 2)}`);
  console.log(`Context: ${JSON.stringify(context, null, 2)}`);
  return {
      statusCode: 200,
      body: JSON.stringify({
          message: 'hello world',
      }),
   };
};
```

5. Tambahkan skrip build ke file package.json. Ini mengonfigurasi esbuild untuk secara otomatis membuat paket penyebaran.zip. Untuk informasi selengkapnya, lihat Membuat skrip di dokumentasi esbuild.

   Linux and MacOS

   ```
   "scripts": {
     "prebuild": "rm -rf dist",
     "build": "esbuild index.ts --bundle --minify --sourcemap --platform=node --
   target=es2020 --outfile=dist/index.js",
     "postbuild": "cd dist && zip -r index.zip index.js*"
   },
   ```

   Windows

   Dalam contoh ini, `"postbuild"` perintah menggunakan utilitas 7zip untuk membuat file.zip Anda. Gunakan utilitas zip Windows pilihan Anda sendiri dan modifikasi perintah seperlunya.

   ```
   "scripts": {
     "prebuild": "del /q dist",
     "build": "esbuild index.ts --bundle --minify --sourcemap --platform=node --
   target=es2020 --outfile=dist/index.js",
     "postbuild": "cd dist && 7z a -tzip index.zip index.js*"
   },
   ```

6. Bangun paketnya.

```
npm run build
```

7. Buat fungsi Lambda menggunakan paket.zip deployment. Ganti teks yang disorot dengan Amazon Resource Name (ARN) dari peran eksekusi Anda.

```
aws lambda create-function --function-name hello-world --runtime "nodejs24.x" --
role arn:aws:iam::123456789012:role/lambda-ex --zip-file "fileb://dist/index.zip"
 --handler index.handler
```

8. Jalankan acara uji untuk mengonfirmasi bahwa fungsi mengembalikan respons berikut. Jika Anda ingin menjalankan fungsi ini menggunakan API Gateway, buat dan konfigurasikan REST API.

```
{
  "statusCode": 200,
  "body": "{\"message\":\"hello world\"}"
}
```

Terapkan TypeScript kode yang ditranspilasikan di Lambda dengan gambar kontainer

Anda dapat menerapkan TypeScript kode Anda ke Fungsi AWS Lambda Acara sebagai gambar Node.js kontainer. AWS menyediakan gambar dasar Node.js untuk membantu Anda membangun gambar kontainer. Gambar dasar ini dimuat sebelumnya dengan runtime bahasa dan komponen lain yang diperlukan untuk menjalankan gambar di Lambda. AWS menyediakan Dockerfile untuk setiap gambar dasar untuk membantu membangun gambar kontainer Anda.

Jika Anda menggunakan citra basis komunitas atau perusahaan swasta, Anda harus menambahkan klien antarmuka Node.js runtime (RIC) ke gambar dasar agar kompatibel dengan Lambda.

Lambda menyediakan emulator antarmuka runtime untuk pengujian lokal. Gambar AWS dasar untuk Node.js menyertakan emulator antarmuka runtime. Jika Anda menggunakan gambar dasar alternatif, seperti gambar Alpine Linux atau Debian, Anda dapat membangun emulator ke dalam gambar Anda atau menginstalnya di mesin lokal Anda.

Menggunakan gambar Node.js dasar untuk membangun dan mengemas kode TypeScript fungsi

Prasyarat

Untuk menyelesaikan langkah-langkah di bagian ini, Anda harus memiliki hal-hal berikut:

- AWS CLI versi 2
- Docker (versi minimum 25.0.0)
- Plugin buildx Docker.
- Node.js 22.x

Membuat gambar dari gambar dasar

Untuk membuat gambar dari sebuah AWS gambar dasar untuk Lambda

1. Pada mesin lokal Anda, buat direktori proyek untuk fungsi baru Anda.
2. Buat Node.js proyek baru dengan `npm` atau manajer paket pilihan Anda.

   ```
   npm init
   ```

3. Tambahkan paket @ types/aws -lambda dan esbuild sebagai dependensi pengembangan. `@types/aws-lambda`Paket berisi definisi tipe untuk Lambda.

```
npm install -D @types/aws-lambda esbuild
```

4. Tambahkan skrip build ke `package.json` file.

```
  "scripts": {
  "build": "esbuild index.ts --bundle --minify --sourcemap --platform=node --
target=es2020 --outfile=dist/index.js"
}
```

5. Buat file baru bernama`index.ts`. Tambahkan kode contoh berikut ke file baru. Ini adalah kode untuk fungsi Lambda. Fungsi mengembalikan `hello world` pesan.

> **Note**
>
> `import`Pernyataan mengimpor definisi tipe dari @ types/aws -lambda. Itu tidak mengimpor paket `aws-lambda` NPM, yang merupakan alat pihak ketiga yang tidak terkait. Untuk informasi selengkapnya, lihat aws-lambda di repositori. DefinitelyTyped GitHub

```
import { Context, APIGatewayProxyResult, APIGatewayEvent } from 'aws-lambda';

export const handler = async (event: APIGatewayEvent, context: Context):
 Promise<APIGatewayProxyResult> => {
    console.log(`Event: ${JSON.stringify(event, null, 2)}`);
    console.log(`Context: ${JSON.stringify(context, null, 2)}`);
    return {
        statusCode: 200,
        body: JSON.stringify({
            message: 'hello world',
        }),
    };
};
```

6. Buat Dockerfile baru dengan konfigurasi berikut:

   - Atur properti `FROM` untuk URI dari gambar dasar.
   - Atur argumen `CMD` untuk menentukan handler fungsi Lambda.

Contoh berikut Dockerfile menggunakan build multi-tahap. Langkah pertama mentransformasikan TypeScript kode ke dalam JavaScript. Langkah kedua menghasilkan gambar kontainer yang hanya berisi JavaScript file dan dependensi produksi.

Perhatikan bahwa contoh Dockerfile tidak menyertakan instruksi USER. Saat Anda menyebarkan gambar kontainer ke Lambda, Lambda secara otomatis mendefinisikan pengguna Linux default dengan izin yang paling tidak memiliki hak istimewa. Ini berbeda dari perilaku Docker standar yang default ke `root` pengguna ketika tidak ada `USER` instruksi yang diberikan.

Example Dockerfile

```
FROM public.ecr.aws/lambda/nodejs:22 as builder
WORKDIR /usr/app
COPY package.json index.ts  ./
RUN npm install
RUN npm run build

FROM public.ecr.aws/lambda/nodejs:22
WORKDIR ${LAMBDA_TASK_ROOT}
COPY --from=builder /usr/app/dist/* ./
CMD ["index.handler"]
```

7. Buat image Docker dengan perintah docker build. Contoh berikut menamai gambar `docker-image` dan memberinya `test` tag. Untuk membuat gambar Anda kompatibel dengan Lambda, Anda harus menggunakan opsi. `--provenance=false`

```
docker buildx build --platform linux/amd64 --provenance=false -t docker-image:test
 .
```

> **Note**
>
> Perintah menentukan `--platform linux/amd64` opsi untuk memastikan bahwa container Anda kompatibel dengan lingkungan eksekusi Lambda terlepas dari arsitektur mesin build Anda. Jika Anda berniat untuk membuat fungsi Lambda menggunakan arsitektur set instruksi ARM64, pastikan untuk mengubah perintah untuk menggunakan opsi sebagai `--platform linux/arm64` gantinya.

(Opsional) Uji gambar secara lokal

1. Mulai gambar Docker dengan perintah docker run. Dalam contoh ini, `docker-image` adalah nama gambar dan `test` tag.

   ```
   docker run --platform linux/amd64 -p 9000:8080 docker-image:test
   ```

   Perintah ini menjalankan gambar sebagai wadah dan membuat titik akhir lokal di`localhost:9000/2015-03-31/functions/function/invocations`.

   > **Note**
   >
   > Jika Anda membuat image Docker untuk arsitektur set instruksi ARM64, pastikan untuk menggunakan `--platform linux/arm64` opsi sebagai gantinya. `--platform linux/amd64`

2. Dari jendela terminal baru, posting acara ke titik akhir lokal.

   Linux/macOS

   Di Linux dan macOS, jalankan perintah berikut: `curl`

   ```
   curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
   ```

   Perintah ini memanggil fungsi dengan peristiwa kosong dan mengembalikan respons. Jika Anda menggunakan kode fungsi Anda sendiri daripada kode fungsi sampel, Anda mungkin ingin memanggil fungsi dengan payload JSON. Contoh:

   ```
   curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d
    '{"payload":"hello world!"}'
   ```

   PowerShell

   Dalam PowerShell, jalankan `Invoke-WebRequest` perintah berikut:

   ```
   Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
   invocations" -Method Post -Body '{}' -ContentType "application/json"
   ```

Perintah ini memanggil fungsi dengan peristiwa kosong dan mengembalikan respons. Jika Anda menggunakan kode fungsi Anda sendiri daripada kode fungsi sampel, Anda mungkin ingin memanggil fungsi dengan payload JSON. Contoh:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
invocations" -Method Post -Body '{"payload":"hello world!"}' -ContentType
 "application/json"
```

3. Dapatkan ID kontainer.

```
docker ps
```

4. Gunakan perintah docker kill untuk menghentikan wadah. Dalam perintah ini, ganti `3766c4ab331c` dengan ID kontainer dari langkah sebelumnya.

```
docker kill 3766c4ab331c
```

Menyebarkan gambar

Untuk mengunggah gambar ke Amazon ECR dan membuat fungsi Lambda

1. Jalankan perintah get-login-password untuk mengautentikasi CLI Docker ke registri Amazon ECR Anda.
   - Tetapkan `--region` nilai ke Wilayah AWS tempat Anda ingin membuat repositori Amazon ECR.
   - Ganti `111122223333` dengan Akun AWS ID Anda.

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --
password-stdin 111122223333.dkr.ecr.us-east-1.amazonaws.com
```

2. Buat repositori di Amazon ECR menggunakan perintah create-repository.

```
aws ecr create-repository --repository-name hello-world --region us-east-1 --image-
scanning-configuration scanOnPush=true --image-tag-mutability MUTABLE
```

> **Note**
>
> Repositori Amazon ECR harus sama Wilayah AWS dengan fungsi Lambda.

Jika berhasil, Anda melihat respons seperti ini:

```
{
    "repository": {
        "repositoryArn": "arn:aws:ecr:us-east-1:111122223333:repository/hello-
world",
        "registryId": "111122223333",
        "repositoryName": "hello-world",
        "repositoryUri": "111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-
world",
        "createdAt": "2023-03-09T10:39:01+00:00",
        "imageTagMutability": "MUTABLE",
        "imageScanningConfiguration": {
            "scanOnPush": true
        },
        "encryptionConfiguration": {
            "encryptionType": "AES256"
        }
    }
}
```

3. Salin `repositoryUri` dari output pada langkah sebelumnya.
4. Jalankan perintah tag docker untuk menandai gambar lokal Anda ke repositori Amazon ECR Anda sebagai versi terbaru. Dalam perintah ini:

   - `docker-image:test`adalah nama dan tag gambar Docker Anda. Ini adalah nama gambar dan tag yang Anda tentukan dalam `docker build` perintah.
   - Ganti <ECRrepositoryUri> dengan `repositoryUri` yang Anda salin. Pastikan untuk menyertakan `:latest` di akhir URI.

```
docker tag docker-image:test <ECRrepositoryUri>:latest
```

Contoh:

```
docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

5. Jalankan perintah docker push untuk menyebarkan gambar lokal Anda ke repositori Amazon ECR. Pastikan untuk menyertakan `:latest` di akhir URI repositori.

```
docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

6. Buat peran eksekusi untuk fungsi tersebut, jika Anda belum memilikinya. Anda memerlukan Nama Sumber Daya Amazon (ARN) dari peran tersebut di langkah berikutnya.
7. Buat fungsi Lambda. Untuk`ImageUri`, tentukan URI repositori dari sebelumnya. Pastikan untuk menyertakan `:latest` di akhir URI.

```
aws lambda create-function \
  --function-name hello-world \
  --package-type Image \
  --code ImageUri=111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
  --role arn:aws:iam::111122223333:role/lambda-ex
```

> **Note**
>
> Anda dapat membuat fungsi menggunakan gambar di AWS akun yang berbeda, selama gambar berada di Wilayah yang sama dengan fungsi Lambda. Untuk informasi selengkapnya, lihat Izin lintas akun Amazon ECR.

8. Memanggil fungsi.

```
aws lambda invoke --function-name hello-world response.json
```

Anda akan melihat tanggapan seperti ini:

```
{
  "ExecutedVersion": "$LATEST",
  "StatusCode": 200
}
```

9. Untuk melihat output dari fungsi, periksa `response.json` file.

Untuk memperbarui kode fungsi, Anda harus membuat gambar lagi, mengunggah gambar baru ke repositori Amazon ECR, dan kemudian menggunakan perintah update-function-code untuk menyebarkan gambar ke fungsi Lambda.

Lambda menyelesaikan tag gambar ke intisari gambar tertentu. Ini berarti bahwa jika Anda mengarahkan tag gambar yang digunakan untuk menyebarkan fungsi ke gambar baru di Amazon ECR, Lambda tidak secara otomatis memperbarui fungsi untuk menggunakan gambar baru.

Untuk menyebarkan gambar baru ke fungsi Lambda yang sama, Anda harus menggunakan perintah update-function-code, meskipun tag gambar di Amazon ECR tetap sama. Dalam contoh berikut, `--publish` opsi membuat versi baru dari fungsi menggunakan gambar kontainer yang diperbarui.

```
aws lambda update-function-code \
  --function-name hello-world \
  --image-uri 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
  --publish
```

Menggunakan objek konteks Lambda untuk mengambil informasi Fungsi Acara TypeScript

Ketika Lambda menjalankan Fungsi Acara Anda, ia meneruskan objek konteks ke handler. Objek ini menyediakan metode dan properti yang memberikan informasi tentang lingkungan invokasi, fungsi, dan eksekusi.

Untuk mengaktifkan pemeriksaan tipe untuk objek konteks, Anda harus menambahkan paket @types/aws -lambda sebagai dependensi pengembangan dan mengimpor jenisnya. `Context` Untuk informasi selengkapnya, lihat Definisi tipe untuk Lambda.

Metode konteks

- `getRemainingTimeInMillis()` – Mengembalikan jumlah milidetik yang tersisa sebelum waktu eksekusi habis.

Properti konteks

- `functionName` – Nama fungsi Lambda.
- `functionVersion` – Versi fungsi.
- `invokedFunctionArn` – Amazon Resource Name (ARN) yang digunakan untuk memicu fungsi. Menunjukkan jika pemicu menyebutkan nomor versi atau alias.
- `memoryLimitInMB` – Jumlah memori yang dialokasikan untuk fungsi tersebut.
- `awsRequestId` – Pengidentifikasi permintaan invokasi.
- `logGroupName` – Grup log untuk fungsi.
- `logStreamName` – Aliran log untuk instans fungsi.
- `identity` – (aplikasi seluler) Informasi tentang identitas Amazon Cognito yang mengesahkan permintaan.
  - `cognitoIdentityId` – Identitas Amazon Cognito terautentikasi.
  - `cognitoIdentityPoolId` – Kumpulan identitas Amazon Cognito yang mengesahkan invokasi.
- `clientContext` – (aplikasi seluler) Konteks klien yang disediakan untuk Lambda oleh aplikasi klien.
  - `client.installation_id`
  - `client.app_title`
  - `client.app_version_name`

  - `client.app_version_code`
  - `client.app_package_name`
  - `env.platform_version`
  - `env.platform`
  - `env.make`
  - `env.model`
  - `env.locale`
  - `Custom` – Nilai kustom yang ditetapkan oleh aplikasi klien.
- `callbackWaitsForEmptyEventLoop`— Secara default (`true`), saat menggunakan penangan fungsi berbasis callback, Lambda menunggu loop peristiwa kosong setelah panggilan balik berjalan sebelum mengakhiri pemanggilan fungsi. Setel `false` untuk mengirim respons dan mengakhiri pemanggilan segera setelah panggilan balik berjalan alih-alih menunggu loop acara kosong. Acara luar biasa terus berjalan selama doa berikutnya. Perhatikan bahwa Lambda mendukung penangan fungsi berbasis callback hanya untuk Node.js 22 runtime dan sebelumnya.

Example file index.ts

Contoh fungsi berikut mencatat informasi konteks dan mengembalikan lokasi log.

> **Note**
>
> Sebelum menggunakan kode ini dalam fungsi Lambda, Anda harus menambahkan paket @types/aws -lambda sebagai dependensi pengembangan. Paket ini berisi definisi tipe untuk Lambda. Lihat informasi yang lebih lengkap di Definisi tipe untuk Lambda.

```
import { Context } from 'aws-lambda';
export const lambdaHandler = async (event: string, context: Context): Promise<string>
 => {
  console.log('Remaining time: ', context.getRemainingTimeInMillis());
  console.log('Function name: ', context.functionName);
  return context.logStreamName;
};
```

Log dan pantau TypeScript Fungsi Acara Lambda

AWS Lambda secara otomatis memonitor fungsi Lambda dan mengirim entri log ke Amazon. CloudWatch Fungsi Lambda Anda dilengkapi dengan grup CloudWatch log Log dan aliran log untuk setiap instance fungsi Anda. Lingkungan runtime Lambda mengirimkan detail tentang setiap pemanggilan dan output lainnya dari kode fungsi Anda ke aliran log. Untuk informasi selengkapnya tentang CloudWatch Log, lihatMengirim log Fungsi Peristiwa Lambda ke Log CloudWatch.

Untuk mengeluarkan log dari kode fungsi Anda, Anda dapat menggunakan metode pada objek konsol. Untuk pencatatan yang lebih rinci, Anda dapat menggunakan pustaka logging apa pun yang menulis ke `stdout` ataustderr.

Bagian-bagian

- Menggunakan alat logging dan pustaka
- Menggunakan Powertools untuk AWS Lambda (TypeScript) dan AWS SAM untuk logging terstruktur
- Menggunakan Powertools untuk AWS Lambda (TypeScript) dan AWS CDK untuk logging terstruktur
- Melihat log di konsol Lambda
- Melihat log di CloudWatch konsol

Menggunakan alat logging dan pustaka

Powertools for AWS Lambda (TypeScript) adalah toolkit pengembang untuk mengimplementasikan praktik terbaik Tanpa Server dan meningkatkan kecepatan pengembang. Utilitas Logger menyediakan logger yang dioptimalkan Lambda yang mencakup informasi tambahan tentang konteks fungsi di semua fungsi Anda dengan output terstruktur sebagai JSON. Gunakan utilitas ini untuk melakukan hal berikut:

- Tangkap bidang kunci dari konteks Lambda, cold start, dan struktur logging output sebagai JSON
- Log peristiwa pemanggilan Lambda saat diinstruksikan (dinonaktifkan secara default)
- Cetak semua log hanya untuk persentase pemanggilan melalui pengambilan sampel log (dinonaktifkan secara default)
- Tambahkan kunci tambahan ke log terstruktur kapan saja
- Gunakan pemformat log kustom (Bring Your Own Formatter) untuk mengeluarkan log dalam struktur yang kompatibel dengan Logging RFC organisasi Anda

Menggunakan Powertools untuk AWS Lambda (TypeScript) dan AWS SAM untuk logging terstruktur

Ikuti langkah-langkah di bawah ini untuk mengunduh, membangun, dan menyebarkan contoh TypeScript aplikasi Hello World dengan modul Powertools for AWS Lambda (TypeScript) terintegrasi menggunakan modul. AWS SAM Aplikasi ini mengimplementasikan backend API dasar dan menggunakan Powertools untuk memancarkan log, metrik, dan jejak. Ini terdiri dari titik akhir Amazon API Gateway dan fungsi Lambda. Saat Anda mengirim permintaan GET ke titik akhir API Gateway, fungsi Lambda memanggil, mengirim log dan metrik menggunakan Format Metrik Tertanam CloudWatch ke, dan mengirimkan jejak ke. AWS X-Ray Fungsi mengembalikan `hello world` pesan.

Prasyarat

Untuk menyelesaikan langkah-langkah di bagian ini, Anda harus memiliki hal-hal berikut:

- Node.js 20 atau lebih
- AWS CLI versi 2
- AWS SAM CLI versi 1.75 atau yang lebih baru. Jika Anda memiliki versi CLI yang lebih lama, lihat Memutakhirkan AWS SAM CLI. AWS SAM

Menyebarkan sampel AWS SAM application

1. Inisialisasi aplikasi menggunakan TypeScript template Hello World.

```
sam init --app-template hello-world-powertools-typescript --name sam-app --package-type Zip --runtime nodejs24.x
```

2. Bangun aplikasi.

```
cd sam-app && sam build
```

3. Terapkan aplikasi.

```
sam deploy --guided
```

4. Ikuti petunjuk di layar. Untuk menerima opsi default yang disediakan dalam pengalaman interaktif, tekan`Enter`.

> **Note**
>
> Karena HelloWorldFunction mungkin tidak memiliki otorisasi yang ditentukan, Apakah ini baik-baik saja? , pastikan untuk masuky.

5. Dapatkan URL aplikasi yang digunakan:

```
aws cloudformation describe-stacks --stack-name sam-app --query
 'Stacks[0].Outputs[?OutputKey==`HelloWorldApi`].OutputValue' --output text
```

6. Memanggil titik akhir API:

```
curl <URL_FROM_PREVIOUS_STEP>
```

Jika berhasil, Anda akan melihat tanggapan ini:

```
{"message":"hello world"}
```

7. Untuk mendapatkan log untuk fungsi tersebut, jalankan log sam. Untuk informasi selengkapnya, lihat Bekerja dengan log di Panduan AWS Serverless Application Model Pengembang.

```
sam logs --stack-name sam-app
```

Output log terlihat seperti ini:

```
2025/01/31/[$LATEST]4d53e8d279824834a1ccd35511a4949c 2025-08-31T09:33:10.552000
 START RequestId: 70693159-7e94-4102-a2af-98a6343fb8fb Version: $LATEST
2025/01/31/[$LATEST]4d53e8d279824834a1ccd35511a4949c 2025-08-31T09:33:10.594000
 2025-08-31T09:33:10.557Z 70693159-7e94-4102-a2af-98a6343fb8fb
 INFO {"_aws":{"Timestamp":1661938390556,"CloudWatchMetrics":
[{"Namespace":"sam-app","Dimensions":[["service"]],"Metrics":
[{"Name":"ColdStart","Unit":"Count"}]}]},"service":"helloWorld","ColdStart":1}
2025/01/31/[$LATEST]4d53e8d279824834a1ccd35511a4949c 2025-08-31T09:33:10.595000
 2025-08-31T09:33:10.595Z 70693159-7e94-4102-a2af-98a6343fb8fb INFO
 {"level":"INFO","message":"This is an INFO log - sending HTTP 200 - hello world
 response","service":"helloWorld","timestamp":"2025-08-31T09:33:10.594Z"}
2025/01/31/[$LATEST]4d53e8d279824834a1ccd35511a4949c 2025-08-31T09:33:10.655000
 2025-08-31T09:33:10.655Z 70693159-7e94-4102-a2af-98a6343fb8fb INFO
 {"_aws":{"Timestamp":1661938390655,"CloudWatchMetrics":[{"Namespace":"sam-
app","Dimensions":[["service"]],"Metrics":[]}]},"service":"helloWorld"}
```

```
2025/01/31/[$LATEST]4d53e8d279824834a1ccd35511a4949c 2025-08-31T09:33:10.754000 END
 RequestId: 70693159-7e94-4102-a2af-98a6343fb8fb
2025/01/31/[$LATEST]4d53e8d279824834a1ccd35511a4949c 2025-08-31T09:33:10.754000
 REPORT RequestId: 70693159-7e94-4102-a2af-98a6343fb8fb Duration: 201.55 ms Billed
 Duration: 202 ms Memory Size: 128 MB Max Memory Used: 66 MB Init Duration: 252.42
 ms
XRAY TraceId: 1-630f2ad5-1de22b6d29a658a466e7ecf5 SegmentId: 567c116658fbf11a
 Sampled: true
```

8. Ini adalah titik akhir API publik yang dapat diakses melalui internet. Kami menyarankan Anda menghapus titik akhir setelah pengujian.

```
sam delete
```

Mengelola retensi log

Grup log tidak terhapus secara otomatis ketika Anda menghapus suatu fungsi. Untuk menghindari penyimpanan log tanpa batas waktu, hapus grup log, atau konfigurasikan periode retensi setelah itu secara CloudWatch otomatis menghapus log. Untuk mengatur penyimpanan log, tambahkan yang berikut ini ke AWS SAM templat Anda:

```
Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function
    Properties:
    # Omitting other properties

  LogGroup:
    Type: AWS::Logs::LogGroup
    Properties:
      LogGroupName: !Sub "/aws/lambda/${HelloWorldFunction}"
      RetentionInDays: 7
```

Menggunakan Powertools untuk AWS Lambda (TypeScript) dan AWS CDK untuk logging terstruktur

Ikuti langkah-langkah di bawah ini untuk mengunduh, membangun, dan menyebarkan contoh TypeScript aplikasi Hello World dengan modul Powertools for AWS Lambda (TypeScript) terintegrasi menggunakan modul. AWS CDK Aplikasi ini mengimplementasikan backend API dasar dan menggunakan Powertools untuk memancarkan log, metrik, dan jejak. Ini terdiri dari titik akhir Amazon API Gateway dan fungsi Lambda. Saat Anda mengirim permintaan GET ke titik akhir

API Gateway, fungsi Lambda memanggil, mengirim log dan metrik menggunakan Format Metrik Tertanam CloudWatch ke, dan mengirimkan jejak ke. AWS X-Ray Fungsi mengembalikan `hello world` pesan.

Prasyarat

Untuk menyelesaikan langkah-langkah di bagian ini, Anda harus memiliki hal-hal berikut:

- Node.js 20 atau lebih
- AWS CLI versi 2
- AWS CDK versi 2
- AWS SAM CLI versi 1.75 atau yang lebih baru. Jika Anda memiliki versi CLI yang lebih lama, lihat Memutakhirkan AWS SAM CLI. AWS SAM

Menyebarkan sampel AWS CDK application

1. Buat direktori proyek untuk aplikasi baru Anda.

```
mkdir hello-world
cd hello-world
```

2. Inisialisasi aplikasi.

```
cdk init app --language typescript
```

3. Tambahkan paket @ types/aws -lambda sebagai dependensi pengembangan.

```
npm install -D @types/aws-lambda
```

4. Instal utilitas Powertools Logger.

```
npm install @aws-lambda-powertools/logger
```

5. Buka direktori lib. Anda akan melihat file bernama hello-world-stack.ts. Buat dua file baru di direktori ini: hello-world.function.ts dan hello-world.ts.
6. Buka hello-world.function.ts dan tambahkan kode berikut ke file. Ini adalah kode untuk fungsi Lambda.

```
import { APIGatewayEvent, APIGatewayProxyResult, Context } from 'aws-lambda';
```

```
import { Logger } from '@aws-lambda-powertools/logger';
const logger = new Logger();

export const handler = async (event: APIGatewayEvent, context: Context):
 Promise<APIGatewayProxyResult> => {
  logger.info('This is an INFO log - sending HTTP 200 - hello world response');
  return {
    statusCode: 200,
    body: JSON.stringify({
      message: 'hello world',
    }),
  };
};
```

7. Buka hello-world.ts dan tambahkan kode berikut ke file. Ini berisi NodejsFunction konstruksi, yang membuat fungsi Lambda, mengonfigurasi variabel lingkungan untuk Powertools, dan menetapkan retensi log menjadi satu minggu. Ini juga mencakup LambdaRestApi konstruksi, yang menciptakan REST API.

```
import { Construct } from 'constructs';
import { NodejsFunction } from 'aws-cdk-lib/aws-lambda-nodejs';
import { LambdaRestApi } from 'aws-cdk-lib/aws-apigateway';
import { RetentionDays } from 'aws-cdk-lib/aws-logs';
import { CfnOutput } from 'aws-cdk-lib';

export class HelloWorld extends Construct {
  constructor(scope: Construct, id: string) {
    super(scope, id);
    const helloFunction = new NodejsFunction(this, 'function', {
      environment: {
        Powertools_SERVICE_NAME: 'helloWorld',
        LOG_LEVEL: 'INFO',
      },
      logRetention: RetentionDays.ONE_WEEK,
    });
    const api = new LambdaRestApi(this, 'apigw', {
      handler: helloFunction,
    });
    new CfnOutput(this, 'apiUrl', {
      exportName: 'apiUrl',
      value: api.url,
    });
  }
```

```
}
```

8. Buka hello-world-stack.ts. Ini adalah kode yang mendefinisikan AWS CDK tumpukan Anda. Ganti kode dengan yang berikut ini:

```
import { Stack, StackProps } from 'aws-cdk-lib';
import { Construct } from 'constructs';
import { HelloWorld } from './hello-world';

export class HelloWorldStack extends Stack {
  constructor(scope: Construct, id: string, props?: StackProps) {
    super(scope, id, props);
    new HelloWorld(this, 'hello-world');
  }
}
```

9. Kembali ke direktori proyek.

```
cd hello-world
```

10. Men-deploy aplikasi Anda.

```
cdk deploy
```

11. Dapatkan URL aplikasi yang digunakan:

```
aws cloudformation describe-stacks --stack-name HelloWorldStack --query
 'Stacks[0].Outputs[?ExportName==`apiUrl`].OutputValue' --output text
```

12. Memanggil titik akhir API:

```
curl <URL_FROM_PREVIOUS_STEP>
```

Jika berhasil, Anda akan melihat tanggapan ini:

```
{"message":"hello world"}
```

13. Untuk mendapatkan log untuk fungsi tersebut, jalankan log sam. Untuk informasi selengkapnya, lihat Bekerja dengan log di Panduan AWS Serverless Application Model Pengembang.

```
sam logs --stack-name HelloWorldStack
```

Output log terlihat seperti ini:

```
2025/01/31/[$LATEST]2ca67f180dcd4d3e88b5d68576740c8e 2025-08-31T14:48:37.047000
 START RequestId: 19ad1007-ff67-40ce-9afe-0af0a9eb512c Version: $LATEST
2025/01/31/[$LATEST]2ca67f180dcd4d3e88b5d68576740c8e 2025-08-31T14:48:37.050000 {
"level": "INFO",
"message": "This is an INFO log - sending HTTP 200 - hello world response",
"service": "helloWorld",
"timestamp": "2025-08-31T14:48:37.048Z",
"xray_trace_id": "1-630f74c4-2b080cf77680a04f2362bcf2"
}
2025/01/31/[$LATEST]2ca67f180dcd4d3e88b5d68576740c8e 2025-08-31T14:48:37.082000 END
 RequestId: 19ad1007-ff67-40ce-9afe-0af0a9eb512c
2025/01/31/[$LATEST]2ca67f180dcd4d3e88b5d68576740c8e 2025-08-31T14:48:37.082000
 REPORT RequestId: 19ad1007-ff67-40ce-9afe-0af0a9eb512c Duration: 34.60 ms Billed
 Duration: 35 ms Memory Size: 128 MB Max Memory Used: 57 MB Init Duration: 173.48
 ms
```

14. Ini adalah titik akhir API publik yang dapat diakses melalui internet. Kami menyarankan Anda menghapus titik akhir setelah pengujian.

```
cdk destroy
```

Melihat log di konsol Lambda

Anda dapat menggunakan konsol Lambda untuk melihat keluaran log setelah Anda menjalankan Fungsi Peristiwa Lambda.

Jika kode Anda dapat diuji dari editor Kode tertanam, Anda menemukan log dalam hasil eksekusi. Saat Anda menggunakan fitur pengujian konsol untuk menjalankan fungsi, Anda menemukan Keluaran Log di bagian Detail.

Melihat log di CloudWatch konsol

Anda dapat menggunakan CloudWatch konsol Amazon untuk melihat log untuk semua pemanggilan Fungsi Peristiwa Lambda.

Untuk melihat log di CloudWatch konsol

1. Buka halaman Grup log di CloudWatch konsol.

2. Pilih grup log untuk fungsi Anda (/aws/lambda/***your-function-name***).
3. Pilih pengaliran log.

Setiap aliran log sesuai dengan instans fungsi Anda. Aliran log muncul saat Anda memperbarui fungsi Lambda, dan saat instance tambahan dibuat untuk menangani pemanggilan bersamaan. Untuk menemukan log untuk pemanggilan tertentu, sebaiknya instrumentasi fungsi Anda dengan. AWS X-Ray X-Ray mencatat detail tentang permintaan dan aliran log dalam jejak.

Menelusuri TypeScript kode di AWS Lambda

Lambda terintegrasi dengan AWS X-Ray untuk membantu Anda melacak, men-debug, dan mengoptimalkan aplikasi Lambda. Anda dapat menggunakan X-Ray untuk melacak permintaan saat melintasi sumber daya dalam aplikasi Anda, yang mungkin termasuk fungsi Lambda dan layanan lainnya. AWS

Untuk mengirim data penelusuran X-Ray, Anda dapat menggunakan salah satu dari tiga pustaka SDK:

- AWS Distro for OpenTelemetry (ADOT) — Distribusi SDK (OTel) yang aman, siap produksi, dan AWS didukung. OpenTelemetry
- AWS X-Ray SDK for Node.js — SDK untuk menghasilkan dan mengirim data jejak ke. X-Ray
- Powertools for AWS Lambda (TypeScript) — Toolkit pengembang untuk mengimplementasikan praktik terbaik Tanpa Server dan meningkatkan kecepatan pengembang.

Setiap SDK menawarkan cara untuk mengirim data telemetri Anda ke layanan. X-Ray Anda kemudian dapat menggunakan X-Ray untuk melihat, memfilter, dan mendapatkan wawasan tentang metrik kinerja aplikasi Anda untuk mengidentifikasi masalah dan peluang untuk pengoptimalan.

> **Important**
>
> The X-Ray dan Powertools untuk AWS Lambda SDK adalah bagian dari solusi instrumentasi terintegrasi ketat yang ditawarkan oleh. AWS Lapisan Lambda ADOT adalah bagian dari standar industri untuk melacak instrumentasi yang mengumpulkan lebih banyak data secara umum, tetapi mungkin tidak cocok untuk semua kasus penggunaan. Anda dapat menerapkan penelusuran ujung ke ujung dalam X-Ray menggunakan salah satu solusi. Untuk mempelajari lebih lanjut tentang memilih di antara keduanya, lihat Memilih antara AWS Distro untuk Telemetri Terbuka dan SDK. X-Ray

Bagian-bagian

- Menggunakan Powertools untuk AWS Lambda (TypeScript) dan AWS SAM untuk menelusuri
- Menggunakan Powertools untuk AWS Lambda (TypeScript) dan AWS CDK untuk menelusuri
- Menafsirkan jejak X-Ray

Menggunakan Powertools untuk AWS Lambda (TypeScript) dan AWS SAM untuk menelusuri

Ikuti langkah-langkah di bawah ini untuk mengunduh, membangun, dan menyebarkan contoh TypeScript aplikasi Hello World dengan modul Powertools for AWS Lambda (TypeScript) terintegrasi menggunakan modul. AWS SAM Aplikasi ini mengimplementasikan backend API dasar dan menggunakan Powertools untuk memancarkan log, metrik, dan jejak. Ini terdiri dari titik akhir Amazon API Gateway dan fungsi Lambda. Saat Anda mengirim permintaan GET ke titik akhir API Gateway, fungsi Lambda memanggil, mengirim log dan metrik menggunakan Format Metrik Tertanam CloudWatch ke, dan mengirimkan jejak ke. AWS X-Ray Fungsi mengembalikan `hello world` pesan.

Prasyarat

Untuk menyelesaikan langkah-langkah di bagian ini, Anda harus memiliki hal-hal berikut:

- Node.js
- AWS CLI versi 2
- AWS SAM CLI versi 1.75 atau yang lebih baru. Jika Anda memiliki versi CLI yang lebih lama, lihat Memutakhirkan AWS SAM CLI. AWS SAM

Menyebarkan sampel AWS SAM application

1. Inisialisasi aplikasi menggunakan TypeScript template Hello World.

```
sam init --app-template hello-world-powertools-typescript --name sam-app --package-type Zip --runtime nodejs24.x --no-tracing
```

2. Bangun aplikasi.

```
cd sam-app && sam build
```

3. Terapkan aplikasi.

```
sam deploy --guided
```

4. Ikuti petunjuk di layar. Untuk menerima opsi default yang disediakan dalam pengalaman interaktif, tekan`Enter`.

> **Note**
>
> Karena HelloWorldFunction mungkin tidak memiliki otorisasi yang ditentukan, Apakah ini baik-baik saja? , pastikan untuk masuky.

5. Dapatkan URL aplikasi yang digunakan:

```
aws cloudformation describe-stacks --stack-name sam-app --query
 'Stacks[0].Outputs[?OutputKey==`HelloWorldApi`].OutputValue' --output text
```

6. Memanggil titik akhir API:

```
curl <URL_FROM_PREVIOUS_STEP>
```

Jika berhasil, Anda akan melihat tanggapan ini:

```
{"message":"hello world"}
```

7. Untuk mendapatkan jejak untuk fungsi tersebut, jalankan jejak sam.

```
sam traces
```

Output jejak terlihat seperti ini:

```
XRay Event [revision 1] at (2023-01-31T11:29:40.527000) with id
 (1-11a2222-111a222222cb33de3b95daf9) and duration (0.483s)
  - 0.425s - sam-app/Prod [HTTP: 200]
    - 0.422s - Lambda [HTTP: 200]
  - 0.406s - sam-app-HelloWorldFunction-Xyzv11a1bcde [HTTP: 200]
  - 0.172s - sam-app-HelloWorldFunction-Xyzv11a1bcde
    - 0.179s - Initialization
    - 0.112s - Invocation
      - 0.052s - ## app.lambdaHandler
        - 0.001s - ### MySubSegment
    - 0.059s - Overhead
```

8. Ini adalah titik akhir API publik yang dapat diakses melalui internet. Kami menyarankan Anda menghapus titik akhir setelah pengujian.

```
sam delete
```

X-Ray tidak melacak semua permintaan ke aplikasi Anda. X-Ray menerapkan algoritma pengambilan sampel untuk memastikan bahwa penelusuran efisien, sambil tetap memberikan sampel yang representatif dari semua permintaan. Tingkat pengambilan sampel adalah 1 permintaan per detik dan 5 persen dari permintaan tambahan. Anda tidak dapat mengonfigurasi laju X-Ray pengambilan sampel untuk fungsi Anda.

Menggunakan Powertools untuk AWS Lambda (TypeScript) dan AWS CDK untuk menelusuri

Ikuti langkah-langkah di bawah ini untuk mengunduh, membangun, dan menyebarkan contoh TypeScript aplikasi Hello World dengan modul Powertools for AWS Lambda (TypeScript) terintegrasi menggunakan modul. AWS CDK Aplikasi ini mengimplementasikan backend API dasar dan menggunakan Powertools untuk memancarkan log, metrik, dan jejak. Ini terdiri dari titik akhir Amazon API Gateway dan fungsi Lambda. Saat Anda mengirim permintaan GET ke titik akhir API Gateway, fungsi Lambda memanggil, mengirim log dan metrik menggunakan Format Metrik Tertanam CloudWatch ke, dan mengirimkan jejak ke. AWS X-Ray Fungsi mengembalikan `hello world` pesan.

Prasyarat

Untuk menyelesaikan langkah-langkah di bagian ini, Anda harus memiliki hal-hal berikut:

- Node.js
- AWS CLI versi 2
- AWS CDK versi 2
- AWS SAM CLI versi 1.75 atau yang lebih baru. Jika Anda memiliki versi CLI yang lebih lama, lihat Memutakhirkan AWS SAM CLI. AWS SAM

Menyebarkan sampel AWS Cloud Development Kit (AWS CDK) application

1. Buat direktori proyek untuk aplikasi baru Anda.

```
mkdir hello-world
cd hello-world
```

2. Inisialisasi aplikasi.

```
cdk init app --language typescript
```

3. Tambahkan paket @ types/aws -lambda sebagai dependensi pengembangan.

```
npm install -D @types/aws-lambda
```

4. Instal utilitas Powertools Tracer.

```
npm install @aws-lambda-powertools/tracer
```

5. Buka direktori lib. Anda akan melihat file bernama hello-world-stack.ts. Buat dua file baru di direktori ini: hello-world.function.ts dan hello-world.ts.

6. Buka hello-world.function.ts dan tambahkan kode berikut ke file. Ini adalah kode untuk fungsi Lambda.

```
import { APIGatewayEvent, APIGatewayProxyResult, Context } from 'aws-lambda';
import { Tracer } from '@aws-lambda-powertools/tracer';
const tracer = new Tracer();

export const handler = async (event: APIGatewayEvent, context: Context):
 Promise<APIGatewayProxyResult> => {
  // Get facade segment created by Lambda
  const segment = tracer.getSegment();

  // Create subsegment for the function and set it as active
  const handlerSegment = segment.addNewSubsegment(`## ${process.env._HANDLER}`);
  tracer.setSegment(handlerSegment);

  // Annotate the subsegment with the cold start and serviceName
  tracer.annotateColdStart();
  tracer.addServiceNameAnnotation();

  // Add annotation for the awsRequestId
  tracer.putAnnotation('awsRequestId', context.awsRequestId);
  // Create another subsegment and set it as active
  const subsegment = handlerSegment.addNewSubsegment('### MySubSegment');
  tracer.setSegment(subsegment);
  let response: APIGatewayProxyResult = {
    statusCode: 200,
    body: JSON.stringify({
      message: 'hello world',
```

```
    }),
  };
  // Close subsegments (the Lambda one is closed automatically)
  subsegment.close(); // (### MySubSegment)
  handlerSegment.close(); // (## index.handler)

  // Set the facade segment as active again (the one created by Lambda)
  tracer.setSegment(segment);
  return response;
};
```

7. Buka hello-world.ts dan tambahkan kode berikut ke file. Ini berisi NodejsFunction konstruksi, yang membuat fungsi Lambda, mengonfigurasi variabel lingkungan untuk Powertools, dan menetapkan retensi log menjadi satu minggu. Ini juga mencakup LambdaRestApi konstruksi, yang menciptakan REST API.

```
import { Construct } from 'constructs';
import { NodejsFunction } from 'aws-cdk-lib/aws-lambda-nodejs';
import { LambdaRestApi } from 'aws-cdk-lib/aws-apigateway';
import { CfnOutput } from 'aws-cdk-lib';
import { Tracing } from 'aws-cdk-lib/aws-lambda';

export class HelloWorld extends Construct {
  constructor(scope: Construct, id: string) {
    super(scope, id);
    const helloFunction = new NodejsFunction(this, 'function', {
      environment: {
        POWERTOOLS_SERVICE_NAME: 'helloWorld',
      },
      tracing: Tracing.ACTIVE,
    });
    const api = new LambdaRestApi(this, 'apigw', {
      handler: helloFunction,
    });
    new CfnOutput(this, 'apiUrl', {
      exportName: 'apiUrl',
      value: api.url,
    });
  }
}
```

8. Buka hello-world-stack.ts. Ini adalah kode yang mendefinisikan AWS CDK tumpukan Anda. Ganti kode dengan yang berikut ini:

```
import { Stack, StackProps } from 'aws-cdk-lib';
import { Construct } from 'constructs';
import { HelloWorld } from './hello-world';

export class HelloWorldStack extends Stack {
  constructor(scope: Construct, id: string, props?: StackProps) {
    super(scope, id, props);
    new HelloWorld(this, 'hello-world');
  }
}
```

9. Men-deploy aplikasi Anda.

```
cd ..
cdk deploy
```

10. Dapatkan URL aplikasi yang digunakan:

```
aws cloudformation describe-stacks --stack-name HelloWorldStack --query
 'Stacks[0].Outputs[?ExportName==`apiUrl`].OutputValue' --output text
```

11. Memanggil titik akhir API:

```
curl <URL_FROM_PREVIOUS_STEP>
```

Jika berhasil, Anda akan melihat tanggapan ini:

```
{"message":"hello world"}
```

12. Untuk mendapatkan jejak untuk fungsi tersebut, jalankan jejak sam.

```
sam traces
```

Output jejak terlihat seperti ini:

```
XRay Event [revision 1] at (2023-01-31T11:50:06.997000) with id
 (1-11a2222-111a222222cb33de3b95daf9) and duration (0.449s)
  - 0.350s - HelloWorldStack-helloworldfunction111A2BCD-Xyzv11a1bcde [HTTP: 200]
  - 0.157s - HelloWorldStack-helloworldfunction111A2BCD-Xyzv11a1bcde
    - 0.169s - Initialization
```

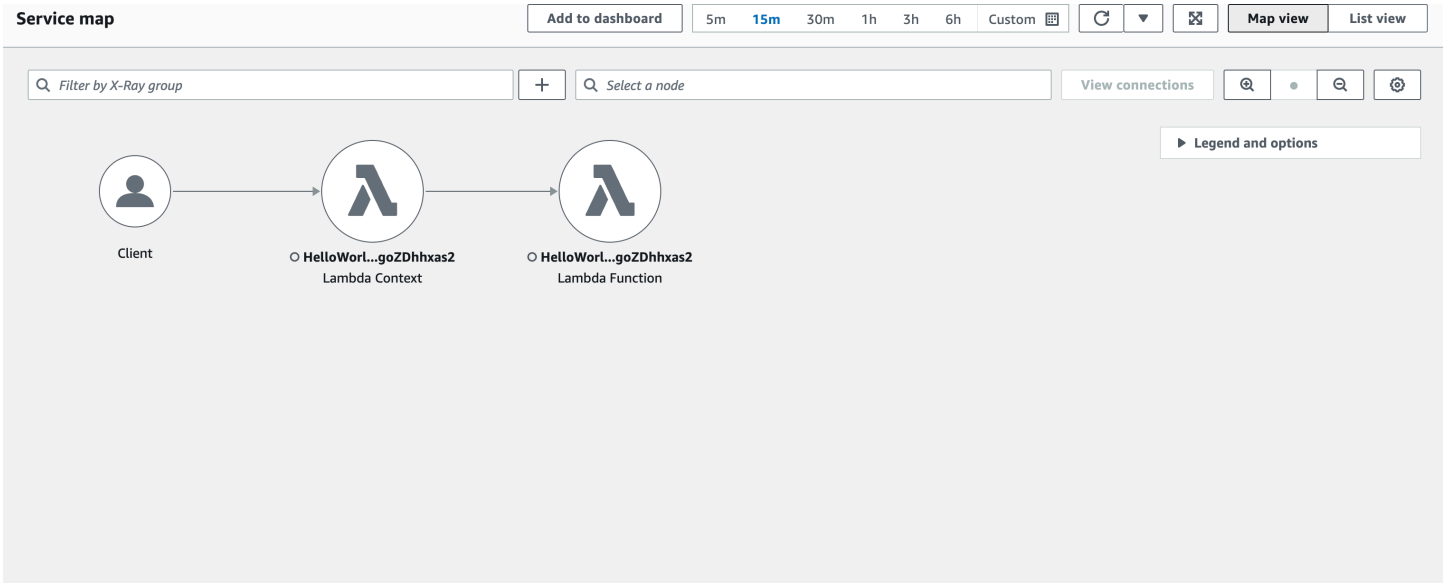
```
    - 0.058s - Invocation
      - 0.055s - ## index.handler
        - 0.000s - ### MySubSegment
    - 0.099s - Overhead
```

13. Ini adalah titik akhir API publik yang dapat diakses melalui internet. Kami menyarankan Anda menghapus titik akhir setelah pengujian.

```
cdk destroy
```

Menafsirkan jejak X-Ray

Setelah mengonfigurasi penelusuran aktif, Anda dapat mengamati permintaan tertentu melalui aplikasi Anda. Peta X-Ray jejak memberikan informasi tentang aplikasi Anda dan semua komponennya. Contoh berikut menunjukkan jejak dari aplikasi sampel:



Membangun Fungsi Acara Lambda dengan Python

Anda dapat menjalankan kode Python di Fungsi AWS Lambda Acara. Lambda menyediakan runtime untuk Python yang menjalankan kode Anda untuk memproses peristiwa. Kode Anda berjalan di lingkungan yang menyertakan SDK for Python (Boto3), dengan AWS Identity and Access Management kredensi dari peran (IAM) yang Anda kelola. Untuk mempelajari lebih lanjut tentang versi SDK yang disertakan dengan runtime Python, lihat. the section called “Runtime-included Versi SDK”

Lambda mendukung runtime Python berikut.

Nama	Pengidentifikasi	Sistem operasi	Tanggal pengusangan	Buat fungsi blok	Pembaruan fungsi blok
Python 3.15	`python3.15`	Amazon Linux 2023	Tidak dijadwalkan	Tidak dijadwalkan	Tidak dijadwalkan
Python 3.14	`python3.14`	Amazon Linux 2023	30 Jun 2029	Jul 31, 2029	Agustus 31, 2029
Python 3.13	`python3.13`	Amazon Linux 2023	30 Jun 2029	Jul 31, 2029	Agustus 31, 2029
Python 3.12	`python3.12`	Amazon Linux 2023	Okt 31, 2028	Nov 30, 2028	10 Jan 2029
Python 3.11	`python3.11`	Amazon Linux 2	30 Jun 2027	Jul 29, 2027	Agustus 31, 2027
Python 3.10	`python3.10`	Amazon Linux 2	Okt 31, 2026	Jul 29, 2027	Agustus 31, 2027

Untuk membuat Fungsi Acara Python

1. Buka Konsol Lambda.
2. Pilih Buat Fungsi Acara.
3. Konfigurasikan pengaturan berikut:
   - Nama fungsi: Masukkan nama untuk fungsi tersebut.
   - Runtime: Pilih Python 3.14.
4. Pilih Buat Fungsi Acara.

Konsol membuat Fungsi Peristiwa Lambda dengan satu file sumber bernama. `lambda_function` Anda dapat mengedit file ini dan menambahkan lebih banyak file di editor kode bawaan. Di bagian DEPLOY, pilih Deploy untuk memperbarui kode fungsi Anda. Kemudian, untuk menjalankan kode Anda, pilih Create test event di bagian TEST EVENTS.

Fungsi Lambda Anda dilengkapi dengan grup CloudWatch log Log. Runtime fungsi mengirimkan detail tentang setiap invokasi ke CloudWatch Logs. Detail tersebut menyampaikan log yang dihasilkan fungsi Anda selama invokasi. Jika fungsi mengembalikan kesalahan, Lambda memformat kesalahan dan mengembalikannya ke pemanggil.

Untuk panduan tentang pengujian fungsi Python Anda, lihat. the section called “Pengujian”

Topik

- Runtime-included Versi SDK
- Fitur Python dinonaktifkan
- Format respons
- Shutdown anggun untuk ekstensi
- Tentukan penangan Fungsi Peristiwa Lambda dengan Python
- Bekerja dengan arsip file.zip untuk Fungsi Acara Python Lambda
- Terapkan Fungsi Acara Lambda Python dengan gambar kontainer
- Bekerja dengan lapisan untuk Fungsi Acara Lambda Python
- Menggunakan objek konteks Lambda untuk mengambil informasi Fungsi Acara Python
- Log dan pantau Fungsi Acara Lambda Python
- AWS Lambda Pengujian Fungsi Acara dengan Python
- Menginstrumentasi kode Python di AWS Lambda

Runtime-included Versi SDK

Versi AWS SDK yang disertakan dalam runtime Python bergantung pada versi runtime dan versi Anda. Wilayah AWS Untuk menemukan versi SDK yang disertakan dalam runtime yang Anda gunakan, buat fungsi dengan kode berikut.

```
import boto3
import botocore

def lambda_handler(event, context):
   print(f'boto3 version: {boto3.__version__}')
   print(f'botocore version: {botocore.__version__}')
```

Fitur Python dinonaktifkan

Tabel berikut mencantumkan fitur Python yang dinonaktifkan di runtime terkelola Lambda dan gambar dasar kontainer untuk Python. Fitur-fitur ini harus diaktifkan ketika runtime Python executable dikompilasi dan tidak dapat diaktifkan dengan menggunakan flag eksekusi waktu. Untuk menggunakan fitur-fitur ini di Lambda, Anda dapat menerapkan build runtime Python Anda sendiri dengan fitur ini diaktifkan, menggunakan gambar kontainer atau runtime kustom.

Fitur Python	Versi Python yang terpengaruh	Status
Just-in-Time (JIT) kompiler	Python 3.13 dan yang lebih baru	Kompiler JIT bersifat eksperimental dan tidak direkomendasikan untuk beban kerja produksi. Oleh karena itu dinonaktifkan di runtime Lambda Python.
Free-threading	Python 3.13 dan yang lebih baru	Threading gratis (opsi untuk menonaktifkan kunci interpreter global) dinonaktifkan di build Lambda Python karena dampak kinerja pada kode single-threaded.

Format respons

Dalam Python 3.12 dan runtime Python yang lebih baru, fungsi mengembalikan karakter Unicode sebagai bagian dari respons JSON mereka. Runtime Python sebelumnya mengembalikan urutan yang diloloskan untuk karakter Unicode dalam tanggapan. Misalnya, dalam Python 3.11, jika Anda mengembalikan string Unicode seperti "こんにちは“, itu lolos dari karakter Unicode dan mengembalikan “\ u3053\ u3093\ u306b\ u3061\ u306f”. Runtime Python 3.12 mengembalikan yang asli "". こんにちは

Menggunakan respons Unicode mengurangi ukuran respons Lambda, membuatnya lebih mudah untuk memasukkan respons yang lebih besar ke dalam ukuran muatan maksimum 6 MB untuk fungsi sinkron. Pada contoh sebelumnya, versi escaped adalah 32 byte — dibandingkan dengan 17 byte dengan string Unicode.

Saat Anda meng-upgrade ke Python 3.12 atau yang lebih baru runtime Python, Anda mungkin perlu menyesuaikan kode Anda untuk memperhitungkan format respons baru. Jika pemanggil mengharapkan Unicode lolos, Anda harus menambahkan kode ke fungsi kembali untuk melarikan

diri dari Unicode secara manual, atau menyesuaikan pemanggil untuk menangani pengembalian Unicode.

Shutdown anggun untuk ekstensi

Python 3.12 dan runtime Python yang lebih baru menawarkan kemampuan shutdown anggun yang ditingkatkan untuk fungsi dengan ekstensi eksternal. Ketika Lambda mematikan lingkungan eksekusi, ia mengirimkan `SIGTERM` sinyal ke runtime dan kemudian `SHUTDOWN` peristiwa ke setiap ekstensi eksternal terdaftar. Anda dapat menangkap `SIGTERM` sinyal dalam fungsi Anda dan membersihkan sumber daya seperti koneksi database yang dibuat oleh fungsi.

Untuk mempelajari lebih lanjut tentang siklus hidup lingkungan eksekusi, lihat. Memahami siklus hidup lingkungan pelaksanaan Lambda Untuk contoh cara menggunakan shutdown anggun dengan ekstensi, lihat Repositori AWS Sampel GitHub .

Tentukan penangan Fungsi Peristiwa Lambda dengan Python

Lambda Event Function handler adalah metode dalam kode fungsi Anda yang memproses peristiwa. Saat fungsi Anda diaktifkan, Lambda menjalankan metode handler. Fungsi Anda berjalan sampai handler mengembalikan respons, keluar, atau waktu habis.

Halaman ini menjelaskan cara bekerja dengan penangan fungsi Lambda dengan Python, termasuk konvensi penamaan, tanda tangan handler yang valid, dan praktik terbaik kode. Halaman ini juga menyertakan contoh Fungsi Peristiwa Lambda Python yang mengambil informasi tentang pesanan, menghasilkan tanda terima file teks, dan menempatkan file ini di bucket Amazon Simple Storage Service (Amazon S3).

Topik

- Contoh kode Fungsi Peristiwa Python Lambda
- Konvensi penamaan handler
- Menggunakan objek acara Lambda
- Mengakses dan menggunakan objek konteks Lambda
- Tanda tangan handler yang valid untuk penangan Python
- Mengembalikan nilai
- Menggunakan AWS SDK untuk Python (Boto3) di handler Anda
- Mengakses variabel lingkungan
- Praktik terbaik kode untuk Fungsi Acara Lambda Python

Contoh kode Fungsi Peristiwa Python Lambda

Contoh kode fungsi Python Lambda berikut mengambil informasi tentang pesanan, menghasilkan tanda terima file teks, dan menempatkan file ini di bucket Amazon S3:

Example Fungsi Acara Lambda Python

```
import json
import os
import logging
import boto3

# Initialize the S3 client outside of the handler
s3_client = boto3.client('s3')
```

```
# Initialize the logger
logger = logging.getLogger()
logger.setLevel("INFO")

def upload_receipt_to_s3(bucket_name, key, receipt_content):
    """Helper function to upload receipt to S3"""

    try:
        s3_client.put_object(
            Bucket=bucket_name,
            Key=key,
            Body=receipt_content
        )
    except Exception as e:
        logger.error(f"Failed to upload receipt to S3: {str(e)}")
        raise

def lambda_handler(event, context):
    """
    Main Lambda handler function
    Parameters:
        event: Dict containing the Lambda function event data
        context: Lambda runtime context
    Returns:
        Dict containing status message
    """
    try:
        # Parse the input event
        order_id = event['Order_id']
        amount = event['Amount']
        item = event['Item']

        # Access environment variables
        bucket_name = os.environ.get('RECEIPT_BUCKET')
        if not bucket_name:
            raise ValueError("Missing required environment variable RECEIPT_BUCKET")

        # Create the receipt content and key destination
        receipt_content = (
            f"OrderID: {order_id}\n"
            f"Amount: ${amount}\n"
            f"Item: {item}"
        )
        key = f"receipts/{order_id}.txt"
```

```
        # Upload the receipt to S3
        upload_receipt_to_s3(bucket_name, key, receipt_content)

        logger.info(f"Successfully processed order {order_id} and stored receipt in S3
bucket {bucket_name}")

        return {
            "statusCode": 200,
            "message": "Receipt processed successfully"
        }

    except Exception as e:
        logger.error(f"Error processing order: {str(e)}")
        raise
```

File ini berisi bagian kode berikut:

- `import`block: Gunakan blok ini untuk menyertakan pustaka yang dibutuhkan fungsi Anda.
- Inisialisasi global klien SDK dan logger: Menyertakan kode inisialisasi di luar handler memanfaatkan penggunaan kembali lingkungan eksekusi untuk meningkatkan kinerja fungsi Anda. Lihat the section called “Praktik terbaik kode untuk Fungsi Acara Lambda Python” untuk mempelajari selengkapnya.
- `def upload_receipt_to_s3(bucket_name, key, receipt_content):`Ini adalah fungsi pembantu yang dipanggil oleh `lambda_handler` fungsi utama.
- `def lambda_handler(event, context):`Ini adalah fungsi handler utama untuk kode Anda, yang berisi logika aplikasi utama Anda. Ketika Lambda memanggil penangan fungsi Anda, runtime Lambda meneruskan dua argumen ke fungsi, objek peristiwa yang berisi data untuk fungsi Anda untuk diproses dan objek konteks yang berisi informasi tentang pemanggilan fungsi.

Konvensi penamaan handler

Nama fungsi handler didefinisikan pada saat Anda membuat fungsi berasal dari:

- Nama file tempat fungsi handler Lambda berada.
- Nama fungsi handler Python.

Dalam contoh di atas, jika file bernama`lambda_function.py`, handler akan ditentukan sebagai`lambda_function.lambda_handler`. Ini adalah nama handler default yang diberikan untuk fungsi yang Anda buat menggunakan konsol Lambda.

Jika Anda membuat fungsi di konsol menggunakan nama file atau nama pengendali fungsi yang berbeda, Anda harus mengedit nama handler default.

Untuk mengubah nama fungsi handler (konsol)

1. Buka halaman Fungsi Acara konsol Lambda dan pilih fungsi Anda.
2. Pilih tab Kode.
3. Gulir ke bawah ke panel pengaturan Runtime dan pilih Edit.
4. Di Handler, masukkan nama baru untuk handler fungsi Anda.
5. Pilih Simpan.

Menggunakan objek acara Lambda

Ketika Lambda memanggil fungsi Anda, ia meneruskan argumen objek peristiwa ke penangan fungsi. Objek JSON adalah format acara yang paling umum untuk Lambda Event Functions. Dalam contoh kode di bagian sebelumnya, fungsi mengharapkan input dalam format berikut:

```
{
    "Order_id": "12345",
    "Amount": 199.99,
    "Item": "Wireless Headphones"
}
```

Jika fungsi Anda dipanggil oleh yang lain Layanan AWS, peristiwa input juga merupakan objek JSON. Format yang tepat dari objek peristiwa tergantung pada layanan yang menjalankan fungsi Anda. Untuk melihat format acara untuk layanan tertentu, lihat halaman yang sesuai di bagian the section called “Mengintegrasikan layanan lain” ini.

Jika peristiwa input dalam bentuk objek JSON, runtime Lambda mengkonversi objek ke kamus Python. Untuk menetapkan nilai dalam input JSON ke variabel dalam kode Anda, gunakan metode kamus Python standar seperti yang diilustrasikan dalam kode contoh.

Anda juga dapat meneruskan data ke dalam fungsi Anda sebagai array JSON, atau sebagai salah satu tipe data JSON valid lainnya. Tabel berikut mendefinisikan bagaimana runtime Python mengkonversi jenis JSON ini.

Jenis data JSON	Tipe data Python
object	kamus (`dict`)
array	daftar (`list`)
number	bilangan bulat (`int`) atau bilangan titik mengambang () `float`
string	string (`str`)
Boolean	Boolean () `bool`
null	NoneType (`NoneType`)

Mengakses dan menggunakan objek konteks Lambda

Objek konteks Lambda berisi informasi tentang fungsi pemanggilan dan lingkungan eksekusi. Lambda meneruskan objek konteks ke fungsi Anda secara otomatis saat dipanggil. Anda dapat menggunakan objek konteks untuk menampilkan informasi tentang pemanggilan fungsi Anda untuk tujuan pemantauan.

Objek konteks adalah kelas Python yang didefinisikan dalam klien antarmuka runtime Lambda. Untuk mengembalikan nilai dari salah satu properti objek konteks, gunakan metode yang sesuai pada objek konteks. Misalnya, cuplikan kode berikut menetapkan nilai `aws_request_id` properti (pengenal untuk permintaan pemanggilan) ke variabel bernama. `request`

```
request = context.aws_request_id
```

Untuk mempelajari lebih lanjut tentang menggunakan objek konteks Lambda, dan untuk melihat daftar lengkap metode dan properti yang tersedia, lihat. the section called “Konteks”

Tanda tangan handler yang valid untuk penangan Python

Saat mendefinisikan fungsi handler Anda dengan Python, fungsi tersebut harus mengambil dua argumen. Argumen pertama adalah objek peristiwa Lambda dan yang kedua adalah objek konteks Lambda. Menurut konvensi, argumen masukan ini biasanya diberi nama `event` dan`context`, tetapi Anda dapat memberi mereka nama apa pun yang Anda inginkan. Jika Anda mendeklarasikan

fungsi handler Anda dengan satu argumen input, Lambda memunculkan kesalahan saat mencoba menjalankan fungsi Anda. Cara paling umum untuk mendeklarasikan fungsi handler dengan Python adalah sebagai berikut:

```
def lambda_handler(event, context):
```

Anda juga dapat menggunakan petunjuk tipe Python dalam deklarasi fungsi Anda, seperti yang ditunjukkan pada contoh berikut:

```
from typing import Dict, Any

def lambda_handler(event: Dict[str, Any], context: Any) -> Dict[str, Any]:
```

Untuk menggunakan AWS pengetikan khusus untuk peristiwa yang dihasilkan oleh other Layanan AWS dan untuk objek konteks, tambahkan `aws-lambda-typing` paket ke paket penerapan fungsi Anda. Anda dapat menginstal pustaka ini di lingkungan pengembangan Anda dengan menjalankan**`pip install aws-lambda-typing`**. Cuplikan kode berikut menunjukkan cara menggunakan petunjuk tipe AWS-specific. Dalam contoh ini, acara yang diharapkan adalah acara Amazon S3.

```
from aws_lambda_typing.events import S3Event
from aws_lambda_typing.context import Context
from typing import Dict, Any

def lambda_handler(event: S3Event, context: Context) -> Dict[str, Any]:
```

Anda tidak dapat menggunakan tipe `async` fungsi Python untuk fungsi handler Anda.

Mengembalikan nilai

Secara opsional, handler dapat mengembalikan nilai, yang harus JSON serializable. Jenis pengembalian umum termasuk`dict`,`list`,`str`,`int`,`float`, dan`bool`.

Apa yang terjadi pada nilai yang dikembalikan tergantung pada jenis pemanggilan dan layanan yang memanggil fungsi. Contoh:

- Jika Anda menggunakan tipe `RequestResponse` pemanggilan untuk memanggil fungsi secara sinkron, Lambda mengembalikan hasil panggilan fungsi Python ke klien yang menjalankan fungsi (dalam respons HTTP terhadap permintaan pemanggilan, diserialisasikan ke dalam JSON).

Misalnya, AWS Lambda konsol menggunakan jenis `RequestResponse` pemanggilan, jadi saat Anda menjalankan fungsi di konsol, konsol akan menampilkan nilai yang dikembalikan.

- Jika penangan mengembalikan objek yang tidak dapat diserialkan `json.dumps`, waktu pengoperasian akan mengembalikan kesalahan. Pengecualian untuk ini adalah jika handler mengembalikan `bytes` objek.
- Jika handler mengembalikan `bytes` objek, runtime melewatinya apa adanya sebagai data biner mentah tanpa serialisasi JSON, menggunakan tipe konten. `application/unknown`
- Jika penangan mengembalikan `None`, sebagaimana fungsi Python tanpa pernyataan `return` yang dilakukan secara implisit, hasil waktu pengoperasian akan mengembalikan `null`.
- Jika Anda menggunakan tipe `Event` pemanggilan (pemanggilan asinkron), nilainya akan dibuang.

Dalam kode contoh, handler mengembalikan kamus Python berikut:

```
{
  "statusCode": 200,
  "message": "Receipt processed successfully"
}
```

Runtime Lambda membuat serial kamus ini dan mengembalikannya ke klien yang memanggil fungsi sebagai string JSON.

> **Note**
>
> Dalam Python 3.9 dan rilis yang lebih baru, Lambda menyertakan Requestid dari pemanggilan dalam respons kesalahan.

Menggunakan AWS SDK untuk Python (Boto3) di handler Anda

Seringkali, Anda akan menggunakan fungsi Anda untuk berinteraksi dengan orang lain Layanan AWS dan sumber daya. Cara termudah untuk berinteraksi dengan sumber daya ini adalah dengan menggunakan file AWS SDK untuk Python (Boto3). Semua runtime Lambda Python yang didukung menyertakan versi SDK for Python (Boto3).

Namun, kami sangat menyarankan agar Anda menyertakan SDK dalam paket penerapan fungsi Anda jika kode Anda perlu menggunakannya. Menyertakan SDK dalam paket penerapan memberi Anda kontrol penuh atas dependensi dan mengurangi risiko masalah ketidaksejajaran versi dengan

pustaka lain. Lihat the section called “Dependensi runtime dengan Python” dan the section called “Kompatibilitas mundur” untuk mempelajari lebih lanjut.

Untuk menggunakan SDK untuk Python (Boto3) dalam fungsi Anda, tambahkan pernyataan berikut ke blok impor di awal kode fungsi Anda:

```
import boto3
```

Gunakan `pip install` perintah untuk menambahkan `boto3` pustaka ke paket penyebaran fungsi Anda. Untuk petunjuk terperinci tentang cara menambahkan dependensi ke paket penerapan.zip, lihat. the section called “Membuat paket penerapan.zip dengan dependensi” Untuk mempelajari selengkapnya tentang menambahkan dependensi ke fungsi yang digunakan sebagai gambar kontainer, lihat atau. the section called “Membuat gambar dari gambar dasar” the section called “Membuat gambar dari gambar dasar alternatif”

Saat menggunakan `boto3` kode Anda, Anda tidak perlu memberikan kredensil apa pun untuk menginisialisasi klien. Misalnya, dalam kode contoh, kami menggunakan baris kode berikut untuk menginisialisasi klien Amazon S3:

```
# Initialize the S3 client outside of the handler
s3_client = boto3.client('s3')
```

Dengan Python, Lambda secara otomatis membuat variabel lingkungan dengan kredensi. `boto3`SDK memeriksa variabel lingkungan fungsi Anda untuk kredensil ini selama inisialisasi.

Mengakses variabel lingkungan

Dalam kode handler Anda, Anda dapat mereferensikan variabel lingkungan dengan menggunakan `os.environ.get` metode ini. Dalam kode contoh, kami mereferensikan variabel `RECEIPT_BUCKET` lingkungan yang ditentukan menggunakan baris kode berikut:

```
# Access environment variables
bucket_name = os.environ.get('RECEIPT_BUCKET')
```

Jangan lupa untuk menyertakan `import os` pernyataan di blok impor di awal kode Anda.

Praktik terbaik kode untuk Fungsi Acara Lambda Python

Patuhi pedoman dalam daftar berikut untuk menggunakan praktik pengkodean terbaik saat membangun fungsi Anda:

- Pisahkan handler Lambda dari logika inti Anda. Ini membuatnya lebih mudah untuk membuat fungsi yang lebih dapat diuji unit. Misalnya, di Python, ini mungkin terlihat seperti:

```
def lambda_handler(event, context):
    foo = event['foo']
    bar = event['bar']
    result = my_lambda_function(foo, bar)

def my_lambda_function(foo, bar):
    // MyLambdaFunction logic here
```

- Kontrol dependensi dalam paket penerapan fungsi Anda. Lingkungan AWS Lambda eksekusi berisi sejumlah pustaka. Untuk runtime Node.js dan Python, ini termasuk SDK. AWS

  Untuk mengaktifkan serangkaian fitur dan pembaruan keamanan terbaru, Lambda memperbarui pustaka ini secara berkala. Pembaruan ini mungkin memperkenalkan perubahan halus pada perilaku fungsi Anda. Untuk memiliki kendali penuh atas dependensi yang digunakan fungsi Anda, kemas semua dependensi Anda dengan paket deployment Anda.
- Minimalkan kompleksitas dependensi Anda. Utamakan memilih kerangka kerja lebih sederhana yang cepat dimuat dalam memulai lingkungan eksekusi.
- Minimalkan ukuran paket penerapan Anda sesuai kebutuhan runtime-nya. Ini mengurangi jumlah waktu yang diperlukan untuk paket penerapan Anda untuk diunduh dan dibongkar sebelum pemanggilan.

Manfaatkan penggunaan kembali lingkungan eksekusi untuk meningkatkan kinerja fungsi Anda. Inisialisasi klien SDK dan koneksi basis data di luar fungsi handler, dan lakukan caching aset statis secara lokal di direktori `/tmp`. Invokasi selanjutnya yang diproses oleh instans yang sama dari fungsi Anda dapat menggunakan kembali sumber daya ini. Ini menghemat biaya dengan mengurangi waktu pengoperasian fungsi.

Untuk menghindari potensi kebocoran data di seluruh invokasi, jangan menggunakan lingkungan eksekusi untuk menyimpan data pengguna, peristiwa, atau informasi lainnya implikasi keamanan. Jika fungsi Anda bergantung pada status yang dapat disenyapkan yang tidak dapat disimpan dalam memori di dalam handler, pertimbangkan untuk membuat fungsi terpisah atau versi terpisah dari fungsi untuk setiap pengguna.

Gunakan arahan keep-alive untuk mempertahankan koneksi yang persisten. Lambda membersihkan koneksi idle dari waktu ke waktu. Mencoba menggunakan ulang koneksi idle saat mengidentifikasi

suatu fungsi akan menyebabkan kesalahan koneksi. Untuk mempertahankan koneksi yang persisten, gunakan arahan tetap aktif yang berkaitan dengan runtime Anda. Sebagai contoh, lihat Menggunakan Kembali Koneksi dengan Keep-Alive in Node.js.

Gunakan variabel lingkungan untuk meneruskan parameter operasional ke fungsi Anda. Misalnya, jika Anda ingin menulis ke bucket Amazon S3 alih-alih melakukan hard-coding nama bucket yang Anda tulis, konfigurasikan nama bucket sebagai variabel lingkungan.

Hindari menggunakan pemanggilan rekursif dalam fungsi Lambda Anda, di mana fungsi memanggil dirinya sendiri atau memulai proses yang dapat memanggil fungsi lagi. Hal ini dapat menyebabkan volume invokasi fungsi yang tidak diinginkan dan peningkatan biaya. Jika Anda melihat volume pemanggilan yang tidak diinginkan, setel konkurensi fungsi cadangan untuk `0` segera membatasi semua pemanggilan ke fungsi, saat Anda memperbarui kode.

Jangan gunakan API non-publik yang tidak terdokumentasi dalam kode fungsi Lambda Anda. Untuk runtime AWS Lambda terkelola, Lambda secara berkala menerapkan pembaruan keamanan dan fungsional ke API internal Lambda. Pembaruan API internal ini mungkin tidak kompatibel ke belakang, yang menyebabkan konsekuensi yang tidak diinginkan seperti kegagalan pemanggilan jika fungsi Anda memiliki ketergantungan pada API non-publik ini. Lihat referensi API untuk daftar API yang tersedia untuk umum.

Tulis kode idempoten. Menulis kode idempoten untuk fungsi Anda memastikan bahwa peristiwa duplikat ditangani dengan cara yang sama. Kode Anda harus memvalidasi peristiwa dengan benar dan menangani peristiwa duplikat dengan anggun. Untuk informasi selengkapnya, lihat Bagaimana cara membuat fungsi Lambda saya idempoten? .

Bekerja dengan arsip file.zip untuk Fungsi Acara Python Lambda

Kode AWS Lambda fungsi Anda terdiri dari file.py yang berisi kode handler fungsi Anda, bersama dengan paket dan modul tambahan yang bergantung pada kode Anda. Untuk menyebarkan kode fungsi ini ke Lambda, Anda menggunakan paket penerapan. Paket ini dapat berupa arsip file.zip atau gambar kontainer. Untuk informasi selengkapnya tentang penggunaan gambar kontainer dengan Python, lihat Menerapkan Fungsi Peristiwa Lambda Python dengan gambar kontainer.

Untuk membuat paket penyebaran Anda sebagai arsip file.zip, Anda dapat menggunakan utilitas arsip file.zip bawaan alat baris perintah Anda, atau utilitas file.zip lainnya seperti 7zip. Contoh yang ditampilkan di bagian berikut mengasumsikan Anda menggunakan `zip` alat baris perintah di lingkungan Linux atau macOS. Untuk menggunakan perintah yang sama di Windows, Anda dapat menginstal Subsistem Windows untuk Linux untuk mendapatkan Windows-integrated versi Ubuntu dan Bash.

Perhatikan bahwa Lambda menggunakan izin file POSIX, jadi Anda mungkin perlu mengatur izin untuk folder paket penyebaran sebelum membuat arsip file.zip.

Topik

- Dependensi runtime dengan Python
- Membuat paket penerapan.zip tanpa dependensi
- Membuat paket penerapan.zip dengan dependensi
- Jalur pencarian ketergantungan dan pustaka yang disertakan runtime
- Menggunakan folder __pycache__
- Membuat paket penerapan.zip dengan pustaka asli
- Membuat dan memperbarui Fungsi Acara Lambda Python menggunakan file.zip

Dependensi runtime dengan Python

Untuk Lambda Event Functions yang menggunakan runtime Python, dependensi dapat berupa paket atau modul Python. Ketika Anda menerapkan fungsi Anda menggunakan arsip.zip, Anda dapat menambahkan dependensi ini ke file.zip Anda dengan kode fungsi Anda atau menggunakan lapisan Lambda. Lapisan adalah file.zip terpisah yang dapat berisi kode tambahan dan konten lainnya. Untuk mempelajari lebih lanjut tentang menggunakan layer Lambda dengan Python, lihat. the section called “Lapisan”

Runtime Lambda Python mencakup dan dependensinya. AWS SDK untuk Python (Boto3) Lambda menyediakan SDK dalam runtime untuk skenario penerapan di mana Anda tidak dapat menambahkan dependensi Anda sendiri. Skenario ini termasuk membuat fungsi di konsol menggunakan editor kode bawaan atau menggunakan fungsi inline in AWS Serverless Application Model (AWS SAM) atau CloudFormation template.

Lambda secara berkala memperbarui pustaka di runtime Python untuk menyertakan pembaruan terbaru dan patch keamanan. Jika fungsi Anda menggunakan versi Boto3 SDK yang disertakan dalam runtime tetapi paket penerapan Anda menyertakan dependensi SDK, ini dapat menyebabkan masalah ketidaksejajaran versi. Misalnya, paket penerapan Anda dapat menyertakan urllib3 dependensi SDK. Saat Lambda memperbarui SDK di runtime, masalah kompatibilitas antara versi baru runtime dan versi urllib3 dalam paket penerapan dapat menyebabkan fungsi Anda gagal.

> **Important**
>
> Untuk mempertahankan kontrol penuh atas dependensi Anda dan untuk menghindari kemungkinan masalah ketidaksejajaran versi, kami sarankan Anda menambahkan semua dependensi fungsi Anda ke paket penerapan Anda, meskipun versinya disertakan dalam runtime Lambda. Ini termasuk Boto3 SDK.

Untuk mengetahui versi SDK for Python (Boto3) mana yang disertakan dalam runtime yang Anda gunakan, lihat. the section called “Runtime-included Versi SDK”

Di bawah model tanggung jawab AWS bersama, Anda bertanggung jawab atas pengelolaan dependensi apa pun dalam paket penerapan fungsi Anda. Ini termasuk menerapkan pembaruan dan patch keamanan. Untuk memperbarui dependensi dalam paket penerapan fungsi Anda, pertama buat file.zip baru dan kemudian unggah ke Lambda. Lihat Membuat paket penerapan.zip dengan dependensi dan Membuat dan memperbarui Fungsi Acara Lambda Python menggunakan file.zip untuk informasi lebih lanjut.

Membuat paket penerapan.zip tanpa dependensi

Jika kode fungsi Anda tidak memiliki dependensi, file.zip Anda hanya berisi file.py dengan kode handler fungsi Anda. Gunakan utilitas zip pilihan Anda untuk membuat file.zip dengan file.py Anda di root. Jika file.py tidak berada di root file.zip Anda, Lambda tidak akan dapat menjalankan kode Anda.

Untuk mempelajari cara menerapkan file.zip Anda untuk membuat fungsi baru atau memperbarui yang sudah ada, lihat. Membuat dan memperbarui Fungsi Acara Lambda Python menggunakan file.zip

Membuat paket penerapan.zip dengan dependensi

Jika kode fungsi Anda bergantung pada paket atau modul tambahan, Anda dapat menambahkan dependensi ini ke file.zip Anda dengan kode fungsi Anda atau menggunakan lapisan Lambda. Petunjuk di bagian ini menunjukkan cara memasukkan dependensi Anda ke dalam paket penerapan.zip Anda. Agar Lambda menjalankan kode Anda, file.py yang berisi kode handler Anda dan semua dependensi fungsi Anda harus diinstal di root file.zip.

Misalkan kode fungsi Anda disimpan dalam file bernama`lambda_function.py`. Contoh perintah CLI berikut membuat file.zip bernama `my_deployment_package.zip` berisi kode fungsi Anda dan dependensinya. Anda dapat menginstal dependensi Anda langsung ke folder di direktori proyek Anda atau menggunakan lingkungan virtual Python.

Untuk membuat paket penyebaran (direktori proyek)

1. Arahkan ke direktori proyek yang berisi file kode `lambda_function.py` sumber Anda. Dalam contoh ini, direktori diberi nama`my_function`.

   ```
   cd my_function
   ```

2. Buat direktori baru bernama paket di mana Anda akan menginstal dependensi Anda.

   ```
   mkdir package
   ```

   Perhatikan bahwa untuk paket penyebaran.zip, Lambda mengharapkan kode sumber Anda dan semua dependensinya berada di root file.zip. Namun, menginstal dependensi langsung di direktori proyek Anda dapat memperkenalkan sejumlah besar file dan folder baru dan membuat navigasi di sekitar IDE Anda menjadi sulit. Anda membuat `package` direktori terpisah di sini untuk menjaga dependensi Anda terpisah dari kode sumber Anda.

3. Instal dependensi Anda di direktori. `package` Contoh di bawah ini menginstal Boto3 SDK dari Python Package Index menggunakan pip. Jika kode fungsi Anda menggunakan paket Python yang telah Anda buat sendiri, simpan di direktori. `package`

   ```
   pip install --target ./package boto3
   ```

4. Buat file.zip dengan pustaka yang diinstal di root.

```
cd package
zip -r ../my_deployment_package.zip .
```

Ini menghasilkan file `my_deployment_package.zip` di direktori proyek Anda.

5. Tambahkan file lambda_function.py ke root dari file.zip

```
cd ..
zip my_deployment_package.zip lambda_function.py
```

File.zip Anda harus memiliki struktur direktori datar, dengan kode handler fungsi Anda dan semua folder dependensi Anda diinstal di root sebagai berikut.

```
my_deployment_package.zip
|- bin
|  |-jp.py
|- boto3
|  |-compat.py
|  |-data
|  |-docs
...
|- lambda_function.py
```

Jika file.py yang berisi kode handler fungsi Anda tidak berada di root file.zip Anda, Lambda tidak dapat menjalankan kode Anda.

Untuk membuat paket penyebaran (lingkungan virtual)

1. Buat dan aktifkan lingkungan virtual di direktori proyek Anda. Dalam contoh ini direktori proyek diberi nama`my_function`.

```
~$ cd my_function
~/my_function$ python3.14 -m venv my_virtual_env
~/my_function$ source ./my_virtual_env/bin/activate
```

2. Instal pustaka yang Anda butuhkan menggunakan pip. Contoh berikut menginstal Boto3 SDK

```
(my_virtual_env) ~/my_function$ pip install boto3
```

3. Gunakan `pip show` untuk menemukan lokasi di lingkungan virtual Anda di mana pip telah menginstal dependensi Anda.

```
(my_virtual_env) ~/my_function$ pip show <package_name>
```

   Folder tempat pip menginstal pustaka Anda mungkin diberi nama atau. `site-packages` `dist-packages` Folder ini mungkin terletak di `lib64/python3.x` direktori `lib/python3.x` or (di mana python3.x mewakili versi Python yang Anda gunakan).

4. Nonaktifkan lingkungan virtual

```
(my_virtual_env) ~/my_function$ deactivate
```

5. Arahkan ke direktori yang berisi dependensi yang Anda instal dengan pip dan buat file.zip di direktori proyek Anda dengan dependensi yang diinstal di root. Dalam contoh ini, pip telah menginstal dependensi Anda di direktori. `my_virtual_env/lib/python3.14/site-packages`

```
~/my_function$ cd my_virtual_env/lib/python3.14/site-packages
~/my_function/my_virtual_env/lib/python3.14/site-packages$ zip -r ../../../../
my_deployment_package.zip .
```

6. Arahkan ke root direktori proyek Anda di mana file.py yang berisi kode handler Anda berada dan tambahkan file itu ke root paket.zip Anda. Dalam contoh ini, file kode fungsi Anda diberi nama`lambda_function.py`.

```
~/my_function/my_virtual_env/lib/python3.14/site-packages$ cd ../../../../
~/my_function$ zip my_deployment_package.zip lambda_function.py
```

Jalur pencarian ketergantungan dan pustaka yang disertakan runtime

Saat Anda menggunakan `import` pernyataan dalam kode Anda, runtime Python mencari direktori di jalur pencariannya hingga menemukan modul atau paket. Secara default, lokasi pertama pencarian runtime adalah direktori tempat paket deployment .zip Anda didekompresi dan dipasang (). `/var/task` Jika Anda menyertakan versi pustaka yang disertakan runtime dalam paket penerapan, versi Anda lebih diutamakan daripada versi yang disertakan dalam runtime. Dependensi dalam paket penerapan Anda juga lebih diutamakan daripada dependensi dalam lapisan.

Saat Anda menambahkan dependensi ke lapisan, Lambda mengekstrak ini `/opt/python/lib/python3.x/site-packages` ke (`python3.x`di mana mewakili versi runtime yang Anda gunakan) atau. `/opt/python` Di jalur pencarian, direktori ini lebih diutamakan daripada direktori yang berisi pustaka yang disertakan runtime dan pustaka yang diinstal pip (dan). `/var/runtime` `/var/lang/lib/python3.x/site-packages` Oleh karena itu, pustaka di lapisan fungsi lebih diutamakan daripada versi yang disertakan dalam runtime.

> **Note**
>
> Dalam runtime terkelola Python 3.11 dan image dasar, AWS SDK dan dependensinya diinstal di direktori. `/var/lang/lib/python3.11/site-packages`

Anda dapat melihat jalur pencarian lengkap untuk fungsi Anda dengan menambahkan cuplikan kode berikut.

```
import sys

search_path = sys.path
print(search_path)
```

> **Note**
>
> Karena dependensi dalam paket atau lapisan penerapan Anda lebih diutamakan daripada pustaka yang disertakan runtime, hal ini dapat menyebabkan masalah ketidaksejajaran versi jika Anda menyertakan dependensi SDK seperti urllib3 dalam paket Anda tanpa menyertakan SDK juga. Jika Anda menerapkan versi dependensi Boto3 Anda sendiri, Anda juga harus menerapkan Boto3 sebagai dependensi dalam paket penerapan Anda. Kami menyarankan Anda mengemas semua dependensi fungsi Anda, bahkan jika versinya disertakan dalam runtime.

Anda juga dapat menambahkan dependensi di folder terpisah di dalam paket.zip Anda. Misalnya, Anda dapat menambahkan versi Boto3 SDK ke folder dalam paket.zip yang disebut. `common` Ketika paket.zip Anda didekompresi dan dipasang, folder ini ditempatkan di dalam direktori. `/var/task` Untuk menggunakan dependensi dari folder dalam paket deployment .zip Anda dalam kode Anda, gunakan pernyataan. `import from` Misalnya, untuk menggunakan versi Boto3 dari folder bernama `common` dalam paket.zip Anda, gunakan pernyataan berikut.

```
from common import boto3
```

Menggunakan folder __pycache__

Kami menyarankan agar Anda tidak menyertakan `__pycache__` folder dalam paket penerapan fungsi Anda. Bytecode Python yang dikompilasi pada mesin build dengan arsitektur atau sistem operasi yang berbeda mungkin tidak kompatibel dengan lingkungan eksekusi Lambda.

Membuat paket penerapan.zip dengan pustaka asli

Jika fungsi Anda hanya menggunakan paket dan modul Python murni, Anda dapat menggunakan `pip install` perintah untuk menginstal dependensi Anda pada mesin build lokal mana pun dan membuat file.zip Anda. Banyak pustaka Python populer, termasuk NumPy dan Panda, bukan Python murni dan berisi kode yang ditulis dalam C atau C ++. Saat menambahkan pustaka yang berisi kode C/C ++ ke paket penerapan, Anda harus membuat paket dengan benar untuk memastikan bahwa paket tersebut kompatibel dengan lingkungan eksekusi Lambda.

Sebagian besar paket yang tersedia di Indeks Paket Python (PyPI) tersedia sebagai “roda” (file.whl). File WHL adalah jenis file ZIP yang berisi distribusi yang dibangun dengan binari pra-kompilasi untuk sistem operasi tertentu dan arsitektur set instruksi. Untuk membuat paket penyebaran Anda kompatibel dengan Lambda, Anda menginstal roda untuk sistem operasi Linux dan arsitektur set instruksi fungsi Anda.

Beberapa paket mungkin hanya tersedia sebagai distribusi sumber. Untuk paket-paket ini, Anda perlu mengkompilasi dan membangun sendiri komponen C/C ++.

Untuk melihat distribusi apa yang tersedia untuk paket yang Anda butuhkan, lakukan hal berikut:

1. Cari nama paket di halaman utama Python Package Index.
2. Pilih versi paket yang ingin Anda gunakan.
3. Pilih Unduh file.

Bekerja dengan distribusi yang dibangun (roda)

Untuk mengunduh roda yang kompatibel dengan Lambda, Anda menggunakan opsi pip `--platform`.

Jika fungsi Anda menggunakan arsitektur set instruksi x86_64, jalankan `pip install` perintah berikut untuk menginstal roda yang kompatibel di direktori Anda. `package` Ganti `--python 3.x` dengan versi runtime Python yang Anda gunakan.

```
pip install \
--platform manylinux2014_x86_64 \
--target=package \
--implementation cp \
--python-version 3.x \
--only-binary=:all: --upgrade \
<package_name>
```

Jika fungsi Anda menggunakan arsitektur set instruksi arm64, jalankan perintah berikut. Ganti `--python 3.x` dengan versi runtime Python yang Anda gunakan.

```
pip install \
--platform manylinux2014_aarch64 \
--target=package \
--implementation cp \
--python-version 3.x \
--only-binary=:all: --upgrade \
<package_name>
```

Bekerja dengan distribusi sumber

Jika paket Anda hanya tersedia sebagai distribusi sumber, Anda perlu membangun pustaka C/C + + sendiri. Untuk membuat paket Anda kompatibel dengan lingkungan eksekusi Lambda, Anda perlu membangunnya di lingkungan yang menggunakan sistem operasi Amazon Linux yang sama. Anda dapat melakukannya dengan membuat paket Anda di instans Linux Amazon Elastic Compute Cloud (Amazon EC2).

Untuk mempelajari cara meluncurkan dan menyambung ke instans Amazon EC2 Linux, lihat Memulai Amazon EC2 di Panduan Pengguna Amazon EC2.

Membuat dan memperbarui Fungsi Acara Lambda Python menggunakan file.zip

Setelah Anda membuat paket.zip deployment, Anda dapat menggunakannya untuk membuat fungsi Lambda baru atau memperbarui yang sudah ada. Anda dapat menerapkan paket.zip Anda menggunakan konsol Lambda, API, AWS Command Line Interface dan Lambda. Anda juga dapat membuat dan memperbarui fungsi Lambda menggunakan AWS Serverless Application Model (AWS SAM) dan. CloudFormation

Ukuran maksimum untuk paket.zip deployment untuk Lambda adalah 250 MB (unzip). Perhatikan bahwa batas ini berlaku untuk ukuran gabungan semua file yang Anda unggah, termasuk lapisan Lambda apa pun.

Runtime Lambda membutuhkan izin untuk membaca file dalam paket deployment Anda. Dalam notasi oktal izin Linux, Lambda membutuhkan 644 izin untuk file yang tidak dapat dieksekusi (rw-r--r--) dan 755 izin (rwxr-xr-x) untuk direktori dan file yang dapat dieksekusi.

Di Linux dan macOS, gunakan `chmod` perintah untuk mengubah izin file pada file dan direktori dalam paket penyebaran Anda. Misalnya, untuk memberikan file yang tidak dapat dieksekusi izin yang benar, jalankan perintah berikut.

```
chmod 644 <filepath>
```

Untuk mengubah izin file di Windows, lihat Mengatur, Melihat, Mengubah, atau Menghapus Izin pada Objek dalam dokumentasi Microsoft Windows.

> **Note**
>
> Jika Anda tidak memberikan Lambda izin yang diperlukan untuk mengakses direktori dalam paket penerapan Anda, Lambda menetapkan izin untuk direktori tersebut ke 755 (rwxr-xr-x).

Membuat dan memperbarui fungsi dengan file.zip menggunakan konsol

Untuk membuat fungsi baru, Anda harus terlebih dahulu membuat fungsi di konsol, lalu mengunggah arsip.zip Anda. Untuk memperbarui fungsi yang ada, buka halaman untuk fungsi Anda, lalu ikuti prosedur yang sama untuk menambahkan file.zip Anda yang diperbarui.

Jika file.zip Anda kurang dari 50MB, Anda dapat membuat atau memperbarui fungsi dengan mengunggah file langsung dari mesin lokal Anda. Untuk file.zip yang lebih besar dari 50MB, Anda harus mengunggah paket Anda ke bucket Amazon S3 terlebih dahulu. Untuk petunjuk tentang cara mengunggah file ke bucket Amazon S3 menggunakan Konsol Manajemen AWS, lihat Memulai Amazon S3. Untuk mengunggah file menggunakan AWS CLI, lihat Memindahkan objek di Panduan AWS CLI Pengguna.

> **Note**
>
> Anda tidak dapat mengubah jenis paket penerapan (.zip atau image kontainer) untuk fungsi yang ada. Misalnya, Anda tidak dapat mengonversi fungsi gambar kontainer untuk menggunakan arsip file.zip. Anda harus membuat fungsi baru.

Untuk membuat fungsi baru (konsol)

1. Buka halaman Fungsi Acara konsol Lambda dan pilih Buat Fungsi Acara.
2. Pilih Tulis dari awal.
3. Di bagian Informasi dasar, lakukan hal berikut:
   a. Untuk nama Fungsi, masukkan nama untuk fungsi Anda.
   b. Untuk Runtime, pilih runtime yang ingin Anda gunakan.
   c. (Opsional) Untuk Arsitektur, pilih arsitektur set instruksi untuk fungsi Anda. Arsitektur defaultnya adalah x86_64. Pastikan bahwa paket deployment .zip untuk fungsi Anda kompatibel dengan arsitektur set instruksi yang Anda pilih.
4. (Opsional) Di bagian Izin, luaskan Ubah peran eksekusi default. Anda dapat membuat peran Eksekusi baru atau menggunakan yang sudah ada.
5. Pilih Buat Fungsi Acara. Lambda menciptakan fungsi dasar 'Hello world' menggunakan runtime yang Anda pilih.

Untuk mengunggah arsip.zip dari mesin lokal Anda (konsol)

1. Di halaman Fungsi Acara konsol Lambda, pilih fungsi yang ingin Anda unggah file.zip.
2. Pilih tab Kode.
3. Di panel Sumber kode, pilih Unggah dari.
4. Pilih file.zip.
5. Untuk mengunggah file.zip, lakukan hal berikut:
   a. Pilih Unggah, lalu pilih file.zip Anda di pemilih file.
   b. Pilih Buka.
   c. Pilih Simpan.

Untuk mengunggah arsip.zip dari bucket Amazon S3 (konsol)

1. Di halaman Fungsi Acara konsol Lambda, pilih fungsi yang ingin Anda unggah file.zip baru.
2. Pilih tab Kode.
3. Di panel Sumber kode, pilih Unggah dari.
4. Pilih lokasi Amazon S3.
5. Rekatkan URL tautan Amazon S3 dari file.zip Anda dan pilih Simpan.

Memperbarui fungsi file.zip menggunakan editor kode konsol

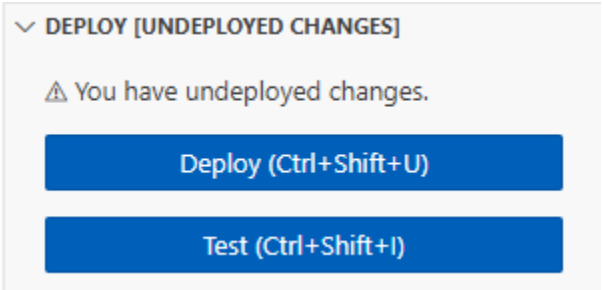
Untuk beberapa fungsi dengan paket penyebaran.zip, Anda dapat menggunakan editor kode bawaan konsol Lambda untuk memperbarui kode fungsi Anda secara langsung. Untuk menggunakan fitur ini, fungsi Anda harus memenuhi kriteria berikut:

- Fungsi Anda harus menggunakan salah satu runtime bahasa yang ditafsirkan (Python,, Node.js atau Ruby)
- Paket penerapan fungsi Anda harus lebih kecil dari 50 MB (unzip).

Kode fungsi untuk fungsi dengan paket penerapan gambar kontainer tidak dapat diedit langsung di konsol.

Untuk memperbarui kode fungsi menggunakan editor kode konsol

1. Buka halaman Fungsi Acara konsol Lambda dan pilih fungsi Anda.
2. Pilih tab Kode.
3. Di panel Sumber kode, pilih file kode sumber Anda dan edit di editor kode terintegrasi.
4. Di bagian DEPLOY, pilih Deploy untuk memperbarui kode fungsi Anda:



Membuat dan memperbarui fungsi dengan file.zip menggunakan AWS CLI

Anda dapat menggunakan AWS CLIuntuk membuat fungsi baru atau memperbarui yang sudah ada menggunakan file.zip. Gunakan perintah create-function dan update-function-code untuk menyebarkan paket.zip Anda. Jika file.zip Anda lebih kecil dari 50MB, Anda dapat mengunggah paket.zip dari lokasi file di mesin build lokal Anda. Untuk file yang lebih besar, Anda harus mengunggah paket.zip Anda dari bucket Amazon S3. Untuk petunjuk tentang cara mengunggah file ke bucket Amazon S3 menggunakan AWS CLI, lihat Memindahkan objek di AWS CLI Panduan Pengguna.

> **Note**
>
> Jika Anda mengunggah file.zip dari bucket Amazon S3 menggunakan AWS CLI bucket, bucket harus berada di lokasi Wilayah AWS yang sama dengan fungsi Anda.

Untuk membuat fungsi baru menggunakan file.zip dengan AWS CLI, Anda harus menentukan yang berikut:

- Nama fungsi Anda (`--function-name`)
- Runtime () `--runtime` fungsi Anda
- Nama Sumber Daya Amazon (ARN) dari peran eksekusi fungsi Anda () `--role`
- Nama metode handler dalam kode fungsi Anda () `--handler`

Anda juga harus menentukan lokasi file.zip Anda. Jika file.zip Anda terletak di folder di mesin build lokal Anda, gunakan `--zip-file` opsi untuk menentukan jalur file, seperti yang ditunjukkan pada perintah contoh berikut.

```
aws lambda create-function --function-name myFunction \
--runtime python3.14 --handler lambda_function.lambda_handler \
--role arn:aws:iam::111122223333:role/service-role/my-lambda-role \
--zip-file fileb://myFunction.zip
```

Untuk menentukan lokasi file.zip di bucket Amazon S3, gunakan opsi seperti `--code` yang ditunjukkan pada perintah contoh berikut. Anda hanya perlu menggunakan `S3ObjectVersion` parameter untuk objek berversi.

```
aws lambda create-function --function-name myFunction \
```

```
--runtime python3.14 --handler lambda_function.lambda_handler \
--role arn:aws:iam::111122223333:role/service-role/my-lambda-role \
--code S3Bucket=amzn-s3-demo-
bucket,S3Key=myFileName.zip,S3ObjectVersion=myObjectVersion
```

Untuk memperbarui fungsi yang ada menggunakan CLI, Anda menentukan nama fungsi Anda menggunakan parameter. `--function-name` Anda juga harus menentukan lokasi file.zip yang ingin Anda gunakan untuk memperbarui kode fungsi Anda. Jika file.zip Anda terletak di folder di mesin build lokal Anda, gunakan `--zip-file` opsi untuk menentukan jalur file, seperti yang ditunjukkan pada perintah contoh berikut.

```
aws lambda update-function-code --function-name myFunction \
--zip-file fileb://myFunction.zip
```

Untuk menentukan lokasi file.zip di bucket Amazon S3, gunakan opsi `--s3-key` dan seperti `--s3-bucket` yang ditunjukkan pada perintah contoh berikut. Anda hanya perlu menggunakan `--s3-object-version` parameter untuk objek berversi.

```
aws lambda update-function-code --function-name myFunction \
--s3-bucket amzn-s3-demo-bucket --s3-key myFileName.zip --s3-object-version myObject
 Version
```

Membuat dan memperbarui fungsi dengan file.zip menggunakan API Lambda

Untuk membuat dan memperbarui fungsi menggunakan arsip file.zip, gunakan operasi API berikut:

- CreateFunction
- UpdateFunctionCode

Membuat dan memperbarui fungsi dengan file.zip menggunakan AWS SAM

The AWS Serverless Application Model (AWS SAM) adalah toolkit yang membantu merampingkan proses membangun dan menjalankan aplikasi tanpa server. AWS Anda menentukan sumber daya untuk aplikasi Anda dalam template YAMAL atau JSON dan menggunakan antarmuka baris AWS SAM perintah (AWS SAM CLI) untuk membangun, mengemas, dan menyebarkan aplikasi Anda. Saat Anda membuat fungsi Lambda dari AWS SAM template, AWS SAM secara otomatis membuat paket penerapan .zip atau gambar kontainer dengan kode fungsi Anda dan dependensi apa pun yang Anda tentukan. Untuk mempelajari lebih lanjut cara menggunakan AWS SAM untuk membangun dan

menerapkan fungsi Lambda, lihat Memulai di Panduan AWS SAMAWS Serverless Application Model Pengembang.

Anda juga dapat menggunakan AWS SAM untuk membuat fungsi Lambda menggunakan arsip file.zip yang ada. Untuk membuat fungsi Lambda menggunakan AWS SAM, Anda dapat menyimpan file.zip di bucket Amazon S3 atau di folder lokal di mesin build Anda. Untuk petunjuk tentang cara mengunggah file ke bucket Amazon S3 menggunakan AWS CLI, lihat Memindahkan objek di AWS CLI Panduan Pengguna.

Dalam AWS SAM template Anda, `AWS::Serverless::Function` sumber daya menentukan fungsi Lambda Anda. Dalam sumber daya ini, atur properti berikut untuk membuat fungsi menggunakan arsip file.zip:

- `PackageType`- diatur ke `Zip`
- `CodeUri`- diatur ke kode fungsi Amazon S3 URI, jalur ke folder lokal, atau objek FunctionCode
- `Runtime`- Setel ke runtime yang Anda pilih

Dengan AWS SAM, jika file.zip Anda lebih besar dari 50MB, Anda tidak perlu mengunggahnya ke bucket Amazon S3 terlebih dahulu. AWS SAM dapat mengunggah paket.zip hingga ukuran maksimum yang diizinkan 250MB (unzip) dari lokasi di mesin build lokal Anda.

Untuk mempelajari selengkapnya tentang penerapan fungsi menggunakan file.zip di AWS SAM, lihat AWS::Serverless: :Fungsi dalam Panduan PengembangAWS SAM .

Membuat dan memperbarui fungsi dengan file.zip menggunakan CloudFormation

Anda dapat menggunakan CloudFormation untuk membuat fungsi Lambda menggunakan arsip file.zip. Untuk membuat fungsi Lambda dari file.zip, Anda harus terlebih dahulu mengunggah file Anda ke bucket Amazon S3. Untuk petunjuk tentang cara mengunggah file ke bucket Amazon S3 menggunakan AWS CLI, lihat Memindahkan objek di AWS CLI Panduan Pengguna.

Untuk Node.js dan runtime Python, Anda juga dapat memberikan kode sumber inline di template Anda. CloudFormation CloudFormation kemudian buat file.zip yang berisi kode Anda saat Anda membangun fungsi Anda.

Menggunakan file.zip yang ada

Dalam CloudFormation template Anda, `AWS::Lambda::Function` sumber daya menentukan fungsi Lambda Anda. Dalam sumber daya ini, atur properti berikut untuk membuat fungsi menggunakan arsip file.zip:

- `PackageType`- Setel ke `Zip`
- `Code`- Masukkan nama bucket Amazon S3 dan nama file.zip di dan bidang `S3Bucket` `S3Key`
- `Runtime`- Setel ke runtime yang Anda pilih

Membuat file.zip dari kode sebaris

Anda dapat mendeklarasikan fungsi sederhana yang ditulis dengan Python atau Node.js inline dalam template. CloudFormation Karena kode disematkan di YAMAL atau JSON, Anda tidak dapat menambahkan dependenis eksternal apa pun ke paket penerapan Anda. Ini berarti fungsi Anda harus menggunakan versi AWS SDK yang disertakan dalam runtime. Persyaratan template, seperti harus melarikan diri dari karakter tertentu, juga mempersulit penggunaan fitur pemeriksaan sintaks dan penyelesaian kode IDE Anda. Ini berarti bahwa template Anda mungkin memerlukan pengujian tambahan. Karena keterbatasan ini, mendeklarasikan fungsi inline paling cocok untuk kode yang sangat sederhana yang tidak sering berubah.

Untuk membuat file.zip dari kode inline for dan runtime Node.js Python, atur properti berikut di sumber daya template Anda: `AWS::Lambda::Function`

- `PackageType`- Setel ke `Zip`
- `Code`- Masukkan kode fungsi Anda di `ZipFile` bidang
- `Runtime`- Setel ke runtime yang Anda pilih

File.zip yang CloudFormation menghasilkan tidak boleh melebihi 4MB. Untuk mempelajari selengkapnya tentang penerapan fungsi menggunakan file.zip di CloudFormation, lihat AWS::Lambda: :Fungsi dalam Panduan Pengguna.CloudFormation

Terapkan Fungsi Acara Lambda Python dengan gambar kontainer

Ada tiga cara untuk membangun image kontainer untuk Fungsi Acara Lambda Python:

- Menggunakan sebuah AWS gambar dasar untuk Python

  Gambar AWS dasar dimuat sebelumnya dengan runtime bahasa, klien antarmuka runtime untuk mengelola interaksi antara Lambda dan kode fungsi Anda, dan emulator antarmuka runtime untuk pengujian lokal.

- Menggunakan sebuah AWS OS-only gambar dasar

  AWS OS-only image dasar berisi distribusi Amazon Linux dan emulator antarmuka runtime. Gambar-gambar ini biasanya digunakan untuk membuat gambar kontainer untuk bahasa yang dikompilasi, seperti Go dan Rust, dan untuk versi bahasa atau bahasa yang Lambda tidak menyediakan gambar dasar, seperti Node.js 19. Anda juga dapat menggunakan gambar OS-only dasar untuk mengimplementasikan runtime kustom. Untuk membuat gambar kompatibel dengan Lambda, Anda harus menyertakan klien antarmuka runtime untuk Python dalam gambar.

- Menggunakan gambar AWS non-dasar

  Anda dapat menggunakan gambar dasar alternatif dari registri kontainer lain, seperti Alpine Linux atau Debian. Anda juga dapat menggunakan gambar kustom yang dibuat oleh organisasi Anda. Untuk membuat gambar kompatibel dengan Lambda, Anda harus menyertakan klien antarmuka runtime untuk Python dalam gambar.

> **Tip**
>
> Untuk mengurangi waktu yang dibutuhkan agar fungsi penampung Lambda menjadi aktif, lihat Menggunakan build multi-tahap dalam dokumentasi Docker. Untuk membuat gambar kontainer yang efisien, ikuti Praktik terbaik untuk menulis Dockerfiles.

Halaman ini menjelaskan cara membuat, menguji, dan menyebarkan gambar kontainer untuk Lambda.

Topik

- AWS gambar dasar untuk Python
- Menggunakan sebuah AWS gambar dasar untuk Python

- Menggunakan gambar dasar alternatif dengan klien antarmuka runtime

AWS gambar dasar untuk Python

AWS menyediakan gambar dasar berikut untuk Python:

Tanda	Waktu berjalan	Sistem operasi	Dockerfile	penghentian
3.15	Python 3.15	Amazon Linux 2023	Dockerfile untuk Python 3.15 aktif GitHub	Tidak dijadwalkan
3.14	Python 3.14	Amazon Linux 2023	Dockerfile untuk Python 3.14 aktif GitHub	30 Jun 2029
3.13	Python 3.13	Amazon Linux 2023	Dockerfile untuk Python 3.13 aktif GitHub	30 Jun 2029
3.12	Python 3.12	Amazon Linux 2023	Dockerfile untuk Python 3.12 aktif GitHub	Okt 31, 2028
3.11	Python 3.11	Amazon Linux 2	Dockerfile untuk Python 3.11 aktif GitHub	30 Jun 2027
3.10	Python 3.10	Amazon Linux 2	Dockerfile untuk Python 3.10 aktif GitHub	Okt 31, 2026

Repositori Amazon ECR: gallery.ecr. aws/lambda/python

Python 3.12 dan gambar dasar yang lebih baru didasarkan pada gambar kontainer minimal Amazon Linux 2023. Gambar dasar Python 3.8-3.11 didasarkan pada gambar Amazon Linux 2. AL2023-based gambar memberikan beberapa keunggulan dibandingkan Amazon Linux 2, termasuk jejak penyebaran yang lebih kecil dan versi pustaka yang diperbarui seperti. `glibc`

AL2023-based gambar menggunakan `microdnf` (symlinked as`dnf`) sebagai manajer paket, bukan`yum`, yang merupakan manajer paket default di Amazon Linux 2. `microdnf`adalah implementasi mandiri dari. `dnf` Untuk daftar paket yang disertakan dalam AL2023-based gambar, lihat kolom Penampung Minimal di Membandingkan paket yang diinstal di Amazon Linux 2023

Container Images. Untuk informasi selengkapnya tentang perbedaan antara AL2023 dan Amazon Linux 2, lihat Memperkenalkan AWS Lambda runtime Amazon Linux 2023 untuk di Blog Komputasi. AWS

> **Note**
>
> Untuk menjalankan AL2023-based gambar secara lokal, termasuk with AWS Serverless Application Model (AWS SAM), Anda harus menggunakan Docker versi 20.10.10 atau yang lebih baru.

Jalur pencarian ketergantungan pada gambar dasar

Saat Anda menggunakan `import` pernyataan dalam kode Anda, runtime Python mencari direktori di jalur pencariannya hingga menemukan modul atau paket. Secara default, runtime mencari `{LAMBDA_TASK_ROOT}` direktori terlebih dahulu. Jika Anda menyertakan versi pustaka yang disertakan runtime dalam gambar Anda, versi Anda akan lebih diutamakan daripada versi yang disertakan dalam runtime.

Langkah-langkah lain di jalur pencarian bergantung pada versi gambar dasar Lambda untuk Python yang Anda gunakan:

- Python 3.11 dan yang lebih baru: Runtime-included pustaka dan pustaka yang diinstal pip diinstal di direktori. `/var/lang/lib/python3.11/site-packages` Direktori ini lebih diutamakan `/var/runtime` di jalur pencarian. Anda dapat mengganti SDK dengan menggunakan pip untuk menginstal versi yang lebih baru. Anda dapat menggunakan pip untuk memverifikasi bahwa SDK yang disertakan runtime dan dependensinya kompatibel dengan paket apa pun yang Anda instal.
- Python 3.8-3.10: Runtime-included pustaka diinstal di direktori. `/var/runtime` Pip-installed pustaka dipasang di `/var/lang/lib/python3.x/site-packages` direktori. `/var/runtime`Direktori lebih diutamakan `/var/lang/lib/python3.x/site-packages` di jalur pencarian.

Anda dapat melihat jalur pencarian lengkap untuk fungsi Anda dengan menambahkan cuplikan kode berikut.

```
import sys

search_path = sys.path
```

```
print(search_path)
```

Menggunakan sebuah AWS gambar dasar untuk Python

Prasyarat

Untuk menyelesaikan langkah-langkah di bagian ini, Anda harus memiliki hal-hal berikut:

- AWS CLI versi 2
- Docker (versi minimum 25.0.0)
- Plugin buildx Docker.
- Python

Membuat gambar dari gambar dasar

Untuk membuat gambar kontainer dari sebuah AWS gambar dasar untuk Python

1. Buat direktori untuk proyek, dan kemudian beralih ke direktori itu.

   ```
   mkdir example
   cd example
   ```

2. Buat file baru bernama`lambda_function.py`. Anda dapat menambahkan kode fungsi contoh berikut ke file untuk pengujian, atau menggunakan kode Anda sendiri.

   Example Fungsi Python

   ```
   import sys
   def handler(event, context):
       return 'Hello from AWS Lambda using Python' + sys.version + '!'
   ```

3. Buat file baru bernama`requirements.txt`. Jika Anda menggunakan kode fungsi sampel dari langkah sebelumnya, Anda dapat membiarkan file kosong karena tidak ada dependensi. Jika tidak, daftarkan setiap pustaka yang diperlukan. Misalnya, inilah tampilan `requirements.txt` Anda jika fungsi Anda menggunakan AWS SDK untuk Python (Boto3):

   Example requirements.txt

   ```
   boto3
   ```

4. Buat Dockerfile baru dengan konfigurasi berikut:

   - Mengatur `FROM` properti ke URI dari gambar dasar.
   - Gunakan perintah COPY untuk menyalin kode fungsi dan dependensi runtime ke`{LAMBDA_TASK_ROOT}`, variabel lingkungan. Lambda-defined
   - Atur `CMD` argumen ke penangan fungsi Lambda.

   Perhatikan bahwa contoh Dockerfile tidak menyertakan instruksi USER. Saat Anda menyebarkan gambar kontainer ke Lambda, Lambda secara otomatis mendefinisikan pengguna Linux default dengan izin yang paling tidak memiliki hak istimewa. Ini berbeda dari perilaku Docker standar yang default ke `root` pengguna ketika tidak ada USER instruksi yang diberikan.

   Example Dockerfile

   ```
   FROM public.ecr.aws/lambda/python:3.12

   # Copy requirements.txt
   COPY requirements.txt ${LAMBDA_TASK_ROOT}

   # Install the specified packages
   RUN pip install -r requirements.txt

   # Copy function code
   COPY lambda_function.py ${LAMBDA_TASK_ROOT}

   # Set the CMD to your handler (could also be done as a parameter override outside
    of the Dockerfile)
   CMD [ "lambda_function.handler" ]
   ```

5. Buat image Docker dengan perintah docker build. Contoh berikut menamai gambar `docker-image` dan memberinya `test` tag. Untuk membuat gambar Anda kompatibel dengan Lambda, Anda harus menggunakan opsi. `--provenance=false`

   ```
   docker buildx build --platform linux/amd64 --provenance=false -t docker-image:test
    .
   ```

> **Note**
>
> Perintah menentukan `--platform linux/amd64` opsi untuk memastikan bahwa container Anda kompatibel dengan lingkungan eksekusi Lambda terlepas dari arsitektur mesin build Anda. Jika Anda bermaksud membuat fungsi Lambda menggunakan arsitektur set instruksi ARM64, pastikan untuk mengubah perintah untuk menggunakan opsi sebagai gantinya. `--platform linux/arm64`

(Opsional) Uji gambar secara lokal

1. Mulai gambar Docker dengan perintah docker run. Dalam contoh ini, `docker-image` adalah nama gambar dan `test` tag.

   ```
   docker run --platform linux/amd64 -p 9000:8080 docker-image:test
   ```

   Perintah ini menjalankan gambar sebagai wadah dan membuat titik akhir lokal di`localhost:9000/2015-03-31/functions/function/invocations`.

   > **Note**
   >
   > Jika Anda membuat image Docker untuk arsitektur set instruksi ARM64, pastikan untuk menggunakan `--platform linux/arm64` opsi sebagai gantinya. `--platform linux/amd64`

2. Dari jendela terminal baru, posting acara ke titik akhir lokal.

   Linux/macOS

   Di Linux dan macOS, jalankan perintah berikut: `curl`

   ```
   curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
   ```

   Perintah ini memanggil fungsi dengan peristiwa kosong dan mengembalikan respons. Jika Anda menggunakan kode fungsi Anda sendiri daripada kode fungsi sampel, Anda mungkin ingin memanggil fungsi dengan payload JSON. Contoh:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d
 '{"payload":"hello world!"}'
```

PowerShell

Dalam PowerShell, jalankan `Invoke-WebRequest` perintah berikut:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
invocations" -Method Post -Body '{}' -ContentType "application/json"
```

Perintah ini memanggil fungsi dengan peristiwa kosong dan mengembalikan respons. Jika Anda menggunakan kode fungsi Anda sendiri daripada kode fungsi sampel, Anda mungkin ingin memanggil fungsi dengan payload JSON. Contoh:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
invocations" -Method Post -Body '{"payload":"hello world!"}' -ContentType
 "application/json"
```

3. Dapatkan ID kontainer.

```
docker ps
```

4. Gunakan perintah docker kill untuk menghentikan kontainer. Dalam perintah ini, ganti `3766c4ab331c` dengan ID kontainer dari langkah sebelumnya.

```
docker kill 3766c4ab331c
```

Menyebarkan gambar

Untuk mengunggah gambar ke Amazon ECR dan membuat fungsi Lambda

1. Jalankan perintah get-login-password untuk mengautentikasi CLI Docker ke registri Amazon ECR Anda.
   - Tetapkan `--region` nilai ke Wilayah AWS tempat Anda ingin membuat repositori Amazon ECR.
   - Ganti `111122223333` dengan Akun AWS ID Anda.

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --
password-stdin 111122223333.dkr.ecr.us-east-1.amazonaws.com
```

2. Buat repositori di Amazon ECR menggunakan perintah create-repository.

```
aws ecr create-repository --repository-name hello-world --region us-east-1 --image-
scanning-configuration scanOnPush=true --image-tag-mutability MUTABLE
```

> **Note**
>
> Repositori Amazon ECR harus sama Wilayah AWS dengan fungsi Lambda.

Jika berhasil, Anda melihat respons seperti ini:

```
{
    "repository": {
        "repositoryArn": "arn:aws:ecr:us-east-1:111122223333:repository/hello-
world",
        "registryId": "111122223333",
        "repositoryName": "hello-world",
        "repositoryUri": "111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-
world",
        "createdAt": "2023-03-09T10:39:01+00:00",
        "imageTagMutability": "MUTABLE",
        "imageScanningConfiguration": {
            "scanOnPush": true
        },
        "encryptionConfiguration": {
            "encryptionType": "AES256"
        }
    }
}
```

3. Salin `repositoryUri` dari output pada langkah sebelumnya.
4. Jalankan perintah tag docker untuk menandai gambar lokal Anda ke repositori Amazon ECR Anda sebagai versi terbaru. Dalam perintah ini:

- `docker-image:test`adalah nama dan tag gambar Docker Anda. Ini adalah nama gambar dan tag yang Anda tentukan dalam `docker build` perintah.
- Ganti `<ECRrepositoryUri>` dengan `repositoryUri` yang Anda salin. Pastikan untuk menyertakan `:latest` di akhir URI.

```
docker tag docker-image:test <ECRrepositoryUri>:latest
```

Contoh:

```
docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

5. Jalankan perintah docker push untuk menyebarkan gambar lokal Anda ke repositori Amazon ECR. Pastikan untuk menyertakan `:latest` di akhir URI repositori.

```
docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

6. Buat peran eksekusi untuk fungsi tersebut, jika Anda belum memilikinya. Anda memerlukan Nama Sumber Daya Amazon (ARN) dari peran tersebut di langkah berikutnya.
7. Buat fungsi Lambda. Untuk`ImageUri`, tentukan URI repositori dari sebelumnya. Pastikan untuk menyertakan `:latest` di akhir URI.

```
aws lambda create-function \
  --function-name hello-world \
  --package-type Image \
  --code ImageUri=111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
  --role arn:aws:iam::111122223333:role/lambda-ex
```

> **Note**
>
> Anda dapat membuat fungsi menggunakan gambar di AWS akun yang berbeda, selama gambar berada di Wilayah yang sama dengan fungsi Lambda. Untuk informasi selengkapnya, lihat Izin lintas akun Amazon ECR.

8. Memanggil fungsi.

```
aws lambda invoke --function-name hello-world response.json
```

Anda akan melihat tanggapan seperti ini:

```
{
  "ExecutedVersion": "$LATEST",
  "StatusCode": 200
}
```

9. Untuk melihat output dari fungsi, periksa `response.json` file.

Untuk memperbarui kode fungsi, Anda harus membuat gambar lagi, mengunggah gambar baru ke repositori Amazon ECR, dan kemudian menggunakan perintah update-function-code untuk menyebarkan gambar ke fungsi Lambda.

Lambda menyelesaikan tag gambar ke intisari gambar tertentu. Ini berarti bahwa jika Anda mengarahkan tag gambar yang digunakan untuk menyebarkan fungsi ke gambar baru di Amazon ECR, Lambda tidak secara otomatis memperbarui fungsi untuk menggunakan gambar baru.

Untuk menyebarkan gambar baru ke fungsi Lambda yang sama, Anda harus menggunakan perintah update-function-code, meskipun tag gambar di Amazon ECR tetap sama. Dalam contoh berikut, `--publish` opsi membuat versi baru dari fungsi menggunakan gambar kontainer yang diperbarui.

```
aws lambda update-function-code \
  --function-name hello-world \
  --image-uri 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
  --publish
```

Menggunakan gambar dasar alternatif dengan klien antarmuka runtime

Jika Anda menggunakan gambar OS-only dasar atau gambar dasar alternatif, Anda harus menyertakan klien antarmuka runtime dalam gambar Anda. Klien antarmuka runtime memperluasAPI runtime, yang mengelola interaksi antara Lambda dan kode fungsi Anda.

Instal klien antarmuka runtime untuk Python menggunakan manajer paket pip:

```
pip install awslambdaric
```

Anda juga dapat mengunduh klien antarmuka runtime Python dari. GitHub

Contoh berikut menunjukkan bagaimana membangun image container untuk Python menggunakan gambar AWS non-dasar. Contoh Dockerfile menggunakan gambar dasar Python resmi. Dockerfile menyertakan klien antarmuka runtime untuk Python.

Prasyarat

Untuk menyelesaikan langkah-langkah di bagian ini, Anda harus memiliki hal-hal berikut:

- AWS CLI versi 2
- Docker (versi minimum 25.0.0)
- Plugin buildx Docker.
- Python

Membuat gambar dari gambar dasar alternatif

Untuk membuat gambar kontainer dari non-AWS gambar dasar

1. Buat direktori untuk proyek, dan kemudian beralih ke direktori itu.

```
mkdir example
cd example
```

2. Buat file baru bernama`lambda_function.py`. Anda dapat menambahkan kode fungsi contoh berikut ke file untuk pengujian, atau menggunakan kode Anda sendiri.

   Example Fungsi Python

```
import sys
def handler(event, context):
    return 'Hello from AWS Lambda using Python' + sys.version + '!'
```

3. Buat file baru bernama`requirements.txt`. Jika Anda menggunakan kode fungsi sampel dari langkah sebelumnya, Anda dapat membiarkan file kosong karena tidak ada dependensi. Jika tidak, daftarkan setiap pustaka yang diperlukan. Misalnya, inilah tampilan `requirements.txt` Anda jika fungsi Anda menggunakan AWS SDK untuk Python (Boto3):

   Example requirements.txt

```
boto3
```

4. Buat Dockerfile baru. Dockerfile berikut menggunakan gambar dasar Python resmi, bukan gambar dasar.AWS Dockerfile menyertakan klien antarmuka runtime, yang membuat gambar kompatibel dengan Lambda. Contoh berikut Dockerfile menggunakan build multi-tahap.

   - Atur `FROM` properti ke gambar dasar.
   - Atur `ENTRYPOINT` ke modul yang Anda inginkan untuk menjalankan wadah Docker saat dimulai. Dalam hal ini, modul adalah klien antarmuka runtime.
   - Atur `CMD` ke penangan fungsi Lambda.

   Perhatikan bahwa contoh Dockerfile tidak menyertakan instruksi USER. Saat Anda menyebarkan gambar kontainer ke Lambda, Lambda secara otomatis mendefinisikan pengguna Linux default dengan izin yang paling tidak memiliki hak istimewa. Ini berbeda dari perilaku Docker standar yang default ke `root` pengguna ketika tidak ada `USER` instruksi yang diberikan.

   Example Dockerfile

```
# Define custom function directory
ARG FUNCTION_DIR="/function"

FROM python:3.12 AS build-image

# Include global arg in this stage of the build
ARG FUNCTION_DIR

# Copy function code
RUN mkdir -p ${FUNCTION_DIR}
COPY . ${FUNCTION_DIR}

# Install the function's dependencies
RUN pip install \
    --target ${FUNCTION_DIR} \
        awslambdaric

# Use a slim version of the base Python image to reduce the final image size
FROM python:3.12-slim

# Include global arg in this stage of the build
ARG FUNCTION_DIR
# Set working directory to function root directory
WORKDIR ${FUNCTION_DIR}
```

```
# Copy in the built dependencies
COPY --from=build-image ${FUNCTION_DIR} ${FUNCTION_DIR}

# Set runtime interface client as default command for the container runtime
ENTRYPOINT [ "/usr/local/bin/python", "-m", "awslambdaric" ]
# Pass the name of the function handler as an argument to the runtime
CMD [ "lambda_function.handler" ]
```

5. Buat image Docker dengan perintah docker build. Contoh berikut menamai gambar `docker-image` dan memberinya `test` tag. Untuk membuat gambar Anda kompatibel dengan Lambda, Anda harus menggunakan opsi. `--provenance=false`

```
docker buildx build --platform linux/amd64 --provenance=false -t docker-image:test
 .
```

> **Note**
>
> Perintah menentukan `--platform linux/amd64` opsi untuk memastikan bahwa container Anda kompatibel dengan lingkungan eksekusi Lambda terlepas dari arsitektur mesin build Anda. Jika Anda bermaksud membuat fungsi Lambda menggunakan arsitektur set instruksi ARM64, pastikan untuk mengubah perintah untuk menggunakan opsi sebagai gantinya. `--platform linux/arm64`

(Opsional) Uji gambar secara lokal

Gunakan emulator antarmuka runtime untuk menguji gambar secara lokal. Anda dapat membangun emulator ke dalam gambar Anda atau menggunakan prosedur berikut untuk menginstalnya di mesin lokal Anda.

Untuk menginstal dan menjalankan emulator antarmuka runtime di mesin lokal Anda

1. Dari direktori proyek Anda, jalankan perintah berikut untuk mengunduh emulator antarmuka runtime (arsitektur x86-64) dari GitHub dan menginstalnya di mesin lokal Anda.

   Linux/macOS

```
mkdir -p ~/.aws-lambda-rie && \
    curl -Lo ~/.aws-lambda-rie/aws-lambda-rie https://github.com/aws/aws-lambda-
runtime-interface-emulator/releases/latest/download/aws-lambda-rie && \
```

```
    chmod +x ~/.aws-lambda-rie/aws-lambda-rie
```

Untuk menginstal emulator arm64, ganti URL GitHub repositori di perintah sebelumnya dengan yang berikut:

```
https://github.com/aws/aws-lambda-runtime-interface-emulator/releases/latest/
download/aws-lambda-rie-arm64
```

PowerShell

```
$dirPath = "$HOME\.aws-lambda-rie"
if (-not (Test-Path $dirPath)) {
    New-Item -Path $dirPath -ItemType Directory
}

$downloadLink = "https://github.com/aws/aws-lambda-runtime-interface-emulator/
releases/latest/download/aws-lambda-rie"
$destinationPath = "$HOME\.aws-lambda-rie\aws-lambda-rie"
Invoke-WebRequest -Uri $downloadLink -OutFile $destinationPath
```

Untuk menginstal emulator arm64, ganti `$downloadLink` dengan yang berikut ini:

```
https://github.com/aws/aws-lambda-runtime-interface-emulator/releases/latest/
download/aws-lambda-rie-arm64
```

2. Mulai gambar Docker dengan perintah docker run. Perhatikan hal-hal berikut:

   - `docker-image`adalah nama gambar dan `test` tag.
   - `/usr/local/bin/python -m awslambdaric lambda_function.handler`adalah `ENTRYPOINT` diikuti oleh `CMD` dari Dockerfile Anda.

   Linux/macOS

```
docker run --platform linux/amd64 -d -v ~/.aws-lambda-rie:/aws-lambda -p
 9000:8080 \
    --entrypoint /aws-lambda/aws-lambda-rie \
    docker-image:test \
        /usr/local/bin/python -m awslambdaric lambda_function.handler
```

PowerShell

```
docker run --platform linux/amd64 -d -v "$HOME\.aws-lambda-rie:/aws-lambda" -p
 9000:8080 `
--entrypoint /aws-lambda/aws-lambda-rie `
docker-image:test `
    /usr/local/bin/python -m awslambdaric lambda_function.handler
```

Perintah ini menjalankan gambar sebagai wadah dan membuat titik akhir lokal di`localhost:9000/2015-03-31/functions/function/invocations`.

> **Note**
>
> Jika Anda membuat image Docker untuk arsitektur set instruksi ARM64, pastikan untuk menggunakan `--platform linux/`*`arm64`* opsi sebagai gantinya. `--platform linux/`*`amd64`*

3. Posting acara ke titik akhir lokal.

Linux/macOS

Di Linux dan macOS, jalankan perintah berikut: `curl`

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
```

Perintah ini memanggil fungsi dengan peristiwa kosong dan mengembalikan respons. Jika Anda menggunakan kode fungsi Anda sendiri daripada kode fungsi sampel, Anda mungkin ingin memanggil fungsi dengan payload JSON. Contoh:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d
 '{"payload":"hello world!"}'
```

PowerShell

Dalam PowerShell, jalankan `Invoke-WebRequest` perintah berikut:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
invocations" -Method Post -Body '{}' -ContentType "application/json"
```

Perintah ini memanggil fungsi dengan peristiwa kosong dan mengembalikan respons. Jika Anda menggunakan kode fungsi Anda sendiri daripada kode fungsi sampel, Anda mungkin ingin memanggil fungsi dengan payload JSON. Contoh:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
invocations" -Method Post -Body '{"payload":"hello world!"}' -ContentType
 "application/json"
```

4. Dapatkan ID kontainer.

```
docker ps
```

5. Gunakan perintah docker kill untuk menghentikan kontainer. Dalam perintah ini, ganti `3766c4ab331c` dengan ID kontainer dari langkah sebelumnya.

```
docker kill 3766c4ab331c
```

Menyebarkan gambar

Untuk mengunggah gambar ke Amazon ECR dan membuat fungsi Lambda

1. Jalankan perintah get-login-password untuk mengautentikasi CLI Docker ke registri Amazon ECR Anda.
   - Tetapkan `--region` nilai ke Wilayah AWS tempat Anda ingin membuat repositori Amazon ECR.
   - Ganti `111122223333` dengan Akun AWS ID Anda.

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --
password-stdin 111122223333.dkr.ecr.us-east-1.amazonaws.com
```

2. Buat repositori di Amazon ECR menggunakan perintah create-repository.

```
aws ecr create-repository --repository-name hello-world --region us-east-1 --image-
scanning-configuration scanOnPush=true --image-tag-mutability MUTABLE
```

> **Note**
>
> Repositori Amazon ECR harus sama Wilayah AWS dengan fungsi Lambda.

Jika berhasil, Anda melihat respons seperti ini:

```
{
    "repository": {
        "repositoryArn": "arn:aws:ecr:us-east-1:111122223333:repository/hello-
world",
        "registryId": "111122223333",
        "repositoryName": "hello-world",
        "repositoryUri": "111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-
world",
        "createdAt": "2023-03-09T10:39:01+00:00",
        "imageTagMutability": "MUTABLE",
        "imageScanningConfiguration": {
            "scanOnPush": true
        },
        "encryptionConfiguration": {
            "encryptionType": "AES256"
        }
    }
}
```

3. Salin `repositoryUri` dari output pada langkah sebelumnya.
4. Jalankan perintah tag docker untuk menandai gambar lokal Anda ke repositori Amazon ECR Anda sebagai versi terbaru. Dalam perintah ini:

   - `docker-image:test`adalah nama dan tag gambar Docker Anda. Ini adalah nama gambar dan tag yang Anda tentukan dalam `docker build` perintah.
   - Ganti <ECRrepositoryUri> dengan `repositoryUri` yang Anda salin. Pastikan untuk menyertakan `:latest` di akhir URI.

```
docker tag docker-image:test <ECRrepositoryUri>:latest
```

Contoh:

```
docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

5. Jalankan perintah docker push untuk menyebarkan gambar lokal Anda ke repositori Amazon ECR. Pastikan untuk menyertakan `:latest` di akhir URI repositori.

```
docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

6. Buat peran eksekusi untuk fungsi tersebut, jika Anda belum memilikinya. Anda memerlukan Nama Sumber Daya Amazon (ARN) dari peran tersebut di langkah berikutnya.
7. Buat fungsi Lambda. Untuk`ImageUri`, tentukan URI repositori dari sebelumnya. Pastikan untuk menyertakan `:latest` di akhir URI.

```
aws lambda create-function \
  --function-name hello-world \
  --package-type Image \
  --code ImageUri=111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
  --role arn:aws:iam::111122223333:role/lambda-ex
```

> **Note**
>
> Anda dapat membuat fungsi menggunakan gambar di AWS akun yang berbeda, selama gambar berada di Wilayah yang sama dengan fungsi Lambda. Untuk informasi selengkapnya, lihat Izin lintas akun Amazon ECR.

8. Memanggil fungsi.

```
aws lambda invoke --function-name hello-world response.json
```

Anda akan melihat tanggapan seperti ini:

```
{
  "ExecutedVersion": "$LATEST",
  "StatusCode": 200
}
```

9. Untuk melihat output dari fungsi, periksa `response.json` file.

Untuk memperbarui kode fungsi, Anda harus membuat gambar lagi, mengunggah gambar baru ke repositori Amazon ECR, dan kemudian menggunakan perintah update-function-code untuk menyebarkan gambar ke fungsi Lambda.

Lambda menyelesaikan tag gambar ke intisari gambar tertentu. Ini berarti bahwa jika Anda mengarahkan tag gambar yang digunakan untuk menyebarkan fungsi ke gambar baru di Amazon ECR, Lambda tidak secara otomatis memperbarui fungsi untuk menggunakan gambar baru.

Untuk menyebarkan gambar baru ke fungsi Lambda yang sama, Anda harus menggunakan perintah update-function-code, meskipun tag gambar di Amazon ECR tetap sama. Dalam contoh berikut, `--publish` opsi membuat versi baru dari fungsi menggunakan gambar kontainer yang diperbarui.

```
aws lambda update-function-code \
  --function-name hello-world \
  --image-uri 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
  --publish
```

Untuk contoh cara membuat gambar Python dari gambar dasar Alpine, lihat Dukungan gambar kontainer untuk Lambda di Blog AWS .

Bekerja dengan lapisan untuk Fungsi Acara Lambda Python

Gunakan lapisan Lambda untuk mengemas kode dan dependensi yang ingin Anda gunakan kembali di beberapa fungsi. Lapisan biasanya berisi dependensi pustaka, runtime kustom, atau file konfigurasi. Membuat layer melibatkan tiga langkah umum:

1. Package konten layer Anda. Ini berarti membuat arsip file.zip yang berisi dependensi yang ingin Anda gunakan dalam fungsi Anda.
2. Buat layer di Lambda.
3. Tambahkan layer ke fungsi Anda.

Topik

- Package konten layer Anda
- Buat layer di Lambda
- Tambahkan layer ke fungsi Anda
- Contoh aplikasi

Package konten layer Anda

Untuk membuat layer, bundel paket Anda ke dalam arsip file.zip yang memenuhi persyaratan berikut:

- Bangun layer menggunakan versi Python yang sama yang Anda rencanakan untuk digunakan untuk fungsi tersebut. Misalnya, jika Anda membangun layer Anda menggunakan Python 3.14, gunakan runtime Python 3.14 untuk fungsi Anda.
- File.zip Anda harus menyertakan `python` direktori di tingkat root.
- Paket di lapisan Anda harus kompatibel dengan Linux. Fungsi Lambda berjalan di Amazon Linux.

Anda dapat membuat lapisan yang berisi pustaka Python pihak ketiga yang diinstal dengan `pip` (seperti `requests` atau`pandas`) atau modul dan paket Python Anda sendiri.

Third-party dependensi

Untuk membuat layer menggunakan paket pip

1. Pilih salah satu metode berikut untuk menginstal `pip` paket ke direktori tingkat atas yang diperlukan (`python/`):

pip install

Untuk paket Python murni (seperti permintaan atau boto3):

```
pip install requests -t python/
```

Beberapa paket Python, seperti NumPy dan Pandas, termasuk komponen C yang dikompilasi. Jika Anda membangun lapisan dengan paket-paket ini di macOS atau Windows, Anda mungkin perlu menggunakan perintah ini untuk menginstal roda: Linux-compatible

```
pip install numpy --platform manylinux2014_x86_64 --only-binary=:all: -t python/
```

Untuk informasi selengkapnya tentang bekerja dengan paket Python yang berisi komponen yang dikompilasi, lihat. Membuat paket penerapan.zip dengan pustaka asli

requirements.txt

Menggunakan `requirements.txt` file membantu Anda mengelola versi paket dan memastikan instalasi yang konsisten.

Example requirements.txt

```
requests==2.31.0
boto3==1.37.34
numpy==1.26.4
```

Jika `requirements.txt` file Anda hanya menyertakan paket Python murni (seperti permintaan atau boto3):

```
pip install -r requirements.txt -t python/
```

Beberapa paket Python, seperti NumPy dan Pandas, termasuk komponen C yang dikompilasi. Jika Anda membangun lapisan dengan paket-paket ini di macOS atau Windows, Anda mungkin perlu menggunakan perintah ini untuk menginstal roda: Linux-compatible

```
pip install -r requirements.txt --platform manylinux2014_x86_64 --only-
binary=:all: -t python/
```

Untuk informasi selengkapnya tentang bekerja dengan paket Python yang berisi komponen yang dikompilasi, lihat. Membuat paket penerapan.zip dengan pustaka asli

2. Zip isi `python` direktori.

   Linux/macOS

   ```
   zip -r layer.zip python/
   ```

   PowerShell

   ```
   Compress-Archive -Path .\python -DestinationPath .\layer.zip
   ```

   Struktur direktori file.zip Anda akan terlihat seperti ini:

   ```
   python/              # Required top-level directory
   ### requests/
   ### boto3/
   ### numpy/
   ### (dependencies of the other packages)
   ```

   > **Note**
   >
   > Jika Anda menggunakan lingkungan virtual Python (venv) untuk menginstal paket, struktur direktori Anda berbeda (misalnya,). `python/lib/python3.x/site-packages` Selama file.zip Anda menyertakan `python` direktori di tingkat root, Lambda dapat menemukan dan mengimpor paket Anda.

Kustom modul Python

Untuk membuat layer menggunakan kode Anda sendiri

1. Buat direktori tingkat atas yang diperlukan untuk lapisan Anda:

   ```
   mkdir python
   ```

2. Buat modul Python Anda di direktori. `python` Modul contoh berikut memvalidasi pesanan dengan mengonfirmasi bahwa mereka berisi informasi yang diperlukan.

   Example modul kustom: validator.py

```
import json

def validate_order(order_data):
    """Validates an order and returns formatted data."""
    required_fields = ['product_id', 'quantity']

    # Check required fields
    missing_fields = [field for field in required_fields if field not in
 order_data]
    if missing_fields:
        raise ValueError(f"Missing required fields: {', '.join(missing_fields)}")

    # Validate quantity
    quantity = order_data['quantity']
    if not isinstance(quantity, int) or quantity < 1:
        raise ValueError("Quantity must be a positive integer")

    # Format and return the validated data
    return {
        'product_id': str(order_data['product_id']),
        'quantity': quantity,
        'shipping_priority': order_data.get('priority', 'standard')
    }

def format_response(status_code, body):
    """Formats the API response."""
    return {
        'statusCode': status_code,
        'body': json.dumps(body)
    }
```

3. Zip isi `python` direktori.

   Linux/macOS

```
zip -r layer.zip python/
```

PowerShell

```
Compress-Archive -Path .\python -DestinationPath .\layer.zip
```

Struktur direktori file.zip Anda akan terlihat seperti ini:

```
python/     # Required top-level directory
### validator.py
```

4. Dalam fungsi Anda, impor dan gunakan modul seperti yang Anda lakukan dengan paket Python apa pun. Contoh:

```
from validator import validate_order, format_response
import json

def lambda_handler(event, context):
    try:
        # Parse the order data from the event body
        order_data = json.loads(event.get('body', '{}'))

        # Validate and format the order
        validated_order = validate_order(order_data)

        return format_response(200, {
            'message': 'Order validated successfully',
            'order': validated_order
        })
    except ValueError as e:
        return format_response(400, {
            'error': str(e)
        })
    except Exception as e:
        return format_response(500, {
            'error': 'Internal server error'
        })
```

Anda dapat menggunakan peristiwa pengujian berikut untuk menjalankan fungsi:

```
{
```

```
    "body": "{\"product_id\": \"ABC123\", \"quantity\": 2, \"priority\": \"express
\"}"
}
```

Tanggapan yang diharapkan:

```
{
  "statusCode": 200,
  "body": "{\"message\": \"Order validated successfully\", \"order\": {\"product_id
\": \"ABC123\", \"quantity\": 2, \"shipping_priority\": \"express\"}}"
}
```

Buat layer di Lambda

Anda dapat mempublikasikan layer Anda menggunakan konsol AWS CLI atau Lambda.

AWS CLI

Jalankan AWS CLI perintah publish-layer-version untuk membuat layer Lambda:

```
aws lambda publish-layer-version --layer-name my-layer --zip-file fileb://layer.zip
 --compatible-runtimes python3.14
```

Parameter runtime yang kompatibel adalah opsional. Saat ditentukan, Lambda menggunakan parameter ini untuk memfilter lapisan di konsol Lambda.

Console

Untuk membuat lapisan (konsol)

1. Buka halaman Fungsi di konsol Lambda.
2. Pilih Buat lapisan.
3. Pilih Unggah file.zip, lalu unggah arsip.zip yang Anda buat sebelumnya.
4. (Opsional) Untuk runtime yang kompatibel, pilih runtime Python yang sesuai dengan versi Python yang Anda gunakan untuk membangun layer Anda.
5. Pilih Buat.

Tambahkan layer ke fungsi Anda

AWS CLI

Untuk melampirkan layer ke fungsi Anda, jalankan perintah AWS CLI update-function-configuration. Untuk `--layers` parameter, gunakan lapisan ARN. ARN harus menentukan versi (misalnya,`arn:aws:lambda:us-east-1:123456789012:layer:my-layer:1`). Untuk informasi selengkapnya, lihat Versi lapisan dan lapisan.

```
aws lambda update-function-configuration --function-name my-function --cli-binary-format raw-in-base64-out --layers "arn:aws:lambda:us-east-1:123456789012:layer:my-layer:1"
```

cli-binary-formatOpsi ini diperlukan jika Anda menggunakan AWS CLI versi 2. Untuk menjadikan ini pengaturan default, jalankan`aws configure set cli-binary-format raw-in-base64-out`. Untuk informasi selengkapnya, lihat opsi baris perintah global yang AWS CLI didukung di Panduan AWS Command Line Interface Pengguna untuk Versi 2.

Console

Untuk menambahkan lapisan ke fungsi

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih fungsinya.
3. Gulir ke bawah ke bagian Layers, lalu pilih Add a layer.
4. Di bawah Pilih layer, pilih Custom layers, dan kemudian pilih layer Anda.

> **Note**
>
> Jika Anda tidak menambahkan runtime yang kompatibel saat membuat layer, layer Anda tidak akan tercantum di sini. Anda dapat menentukan lapisan ARN sebagai gantinya.

5. Pilih Tambahkan.

Contoh aplikasi

Untuk contoh lebih lanjut tentang cara menggunakan layer Lambda, lihat contoh aplikasi layer-python di repositori Panduan Pengembang. AWS Lambda GitHub Aplikasi ini mencakup dua lapisan yang

berisi pustaka Python. Setelah membuat layer, Anda dapat menerapkan dan memanggil fungsi yang sesuai untuk mengonfirmasi bahwa lapisan berfungsi seperti yang diharapkan.

Menggunakan objek konteks Lambda untuk mengambil informasi Fungsi Acara Python

Ketika Lambda menjalankan Fungsi Acara Anda, ia meneruskan objek konteks ke handler. Objek ini menyediakan metode dan properti yang memberikan informasi tentang lingkungan invokasi, fungsi, dan eksekusi. Untuk informasi selengkapnya tentang bagaimana objek konteks diteruskan ke handler fungsi, lihat Tentukan penangan Fungsi Peristiwa Lambda dengan Python.

Metode konteks

- `get_remaining_time_in_millis` – Mengembalikan jumlah milidetik yang tersisa sebelum waktu eksekusi habis.

Properti konteks

- `function_name` – Nama fungsi Lambda.
- `function_version` – Versi fungsi.
- `invoked_function_arn` – Amazon Resource Name (ARN) yang digunakan untuk memicu fungsi. Menunjukkan jika pemicu menyebutkan nomor versi atau alias.
- `memory_limit_in_mb` – Jumlah memori yang dialokasikan untuk fungsi tersebut.
- `aws_request_id` – Pengidentifikasi permintaan invokasi.
- `log_group_name` – Grup log untuk fungsi.
- `log_stream_name` – Aliran log untuk instans fungsi.
- `identity` – (aplikasi seluler) Informasi tentang identitas Amazon Cognito yang mengesahkan permintaan.
  - `cognito_identity_id` – Identitas Amazon Cognito terautentikasi.
  - `cognito_identity_pool_id` – Kumpulan identitas Amazon Cognito yang mengesahkan invokasi.
- `client_context` – (aplikasi seluler) Konteks klien yang disediakan untuk Lambda oleh aplikasi klien.
  - `client.installation_id`
  - `client.app_title`
  - `client.app_version_name`
  - `client.app_version_code`
  - `client.app_package_name`

- `custom` – `dict` nilai khusus yang diatur oleh aplikasi klien seluler.
- `env`— Informasi lingkungan yang disediakan oleh AWS SDK. `dict`

Powertools untuk Lambda (Python) memberikan definisi antarmuka untuk objek konteks Lambda. Anda dapat menggunakan definisi antarmuka untuk petunjuk tipe, atau untuk memeriksa lebih lanjut struktur objek konteks Lambda. Untuk definisi antarmuka, lihat lambda_context.py di repositori powertools-lambda-python pada. GitHub

Contoh berikut menunjukkan fungsi penangan yang mencatat informasi konteks.

Example handler.py

```
import time

def lambda_handler(event, context):
    print("Lambda function ARN:", context.invoked_function_arn)
    print("CloudWatch log stream name:", context.log_stream_name)
    print("CloudWatch log group name:",  context.log_group_name)
    print("Lambda Request ID:", context.aws_request_id)
    print("Lambda function memory limits in MB:", context.memory_limit_in_mb)
    # We have added a 1 second delay so you can see the time remaining in
 get_remaining_time_in_millis.
    time.sleep(1)
    print("Lambda time remaining in MS:", context.get_remaining_time_in_millis())
```

Selain opsi yang tercantum di atas, Anda juga dapat menggunakan AWS X-Ray SDK Menginstrumentasi kode Python di AWS Lambda untuk mengidentifikasi jalur kode penting, melacak kinerjanya, dan menangkap data untuk dianalisis.

Log dan pantau Fungsi Acara Lambda Python

AWS Lambda secara otomatis memonitor fungsi Lambda dan mengirim entri log ke Amazon. CloudWatch Fungsi Lambda Anda dilengkapi dengan grup CloudWatch log Log dan aliran log untuk setiap instance fungsi Anda. Lingkungan runtime Lambda mengirimkan detail tentang setiap pemanggilan dan output lainnya dari kode fungsi Anda ke aliran log. Untuk informasi selengkapnya tentang CloudWatch Log, lihatMengirim log Fungsi Peristiwa Lambda ke Log CloudWatch.

Untuk mengeluarkan log dari kode fungsi Anda, Anda dapat menggunakan `logging`modul bawaan. Untuk entri yang lebih rinci, Anda dapat menggunakan pustaka logging apa pun yang menulis ke `stdout` ataustderr.

Mencetak ke log

Untuk mengirim output dasar ke log, Anda dapat menggunakan `print` metode dalam fungsi Anda. Contoh berikut mencatat nilai-nilai grup CloudWatch log Log dan aliran, dan objek acara.

Perhatikan bahwa jika fungsi Anda mengeluarkan log menggunakan pernyataan `print` Python, Lambda hanya dapat mengirim output CloudWatch log ke Log dalam format teks biasa. Untuk menangkap log di JSON terstruktur, Anda perlu menggunakan pustaka logging yang didukung. Untuk informasi selengkapnya, lihat the section called “Menggunakan kontrol logging lanjutan Lambda dengan Python”.

Example lambda_function.py

```
import os
def lambda_handler(event, context):
    print('## ENVIRONMENT VARIABLES')
    print(os.environ['AWS_LAMBDA_LOG_GROUP_NAME'])
    print(os.environ['AWS_LAMBDA_LOG_STREAM_NAME'])
    print('## EVENT')
    print(event)
```

Example output log

```
START RequestId: 8f507cfc-xmpl-4697-b07a-ac58fc914c95 Version: $LATEST
## ENVIRONMENT VARIABLES
/aws/lambda/my-function
2025/08/31/[$LATEST]3893xmpl7fac4485b47bb75b671a283c
## EVENT
{'key': 'value'}
END RequestId: 8f507cfc-xmpl-4697-b07a-ac58fc914c95
```

```
REPORT RequestId: 8f507cfc-xmpl-4697-b07a-ac58fc914c95  Duration: 15.74 ms  Billed
 Duration: 147 ms Memory Size: 128 MB Max Memory Used: 56 MB  Init Duration: 130.49 ms
XRAY TraceId: 1-5e34a614-10bdxmplf1fb44f07bc535a1   SegmentId: 07f5xmpl2d1f6f85
 Sampled: true
```

Runtime Python melakukan log baris `START`, `END`, dan `REPORT` untuk setiap invokasi. `REPORT`Baris tersebut mencakup data berikut:

Laporkan bidang data baris

- RequestId – ID permintaan unik untuk invokasi.
- Durasi – Jumlah waktu yang digunakan oleh metode handler fungsi Anda gunakan untuk memproses peristiwa.
- Durasi yang Ditagih – Jumlah waktu yang ditagihkan untuk invokasi.
- Ukuran Memori – Jumlah memori yang dialokasikan untuk fungsi.
- Memori Maks yang Digunakan – Jumlah memori yang digunakan oleh fungsi. Saat pemanggilan berbagi lingkungan eksekusi, Lambda melaporkan memori maksimum yang digunakan di semua pemanggilan. Perilaku ini dapat menghasilkan nilai yang dilaporkan lebih tinggi dari yang diharapkan.
- Durasi Init – Untuk permintaan pertama yang dilayani, lama waktu yang diperlukan runtime untuk memuat fungsi dan menjalankan kode di luar metode handler.
- XRAY TraceId — Untuk permintaan yang dilacak, ID AWS X-Ray jejak.
- SegmentId— Untuk permintaan yang dilacak, ID X-Ray segmen.
- Diambil Sampel – Untuk permintaan yang dilacak, hasil pengambilan sampel.

Menggunakan pustaka logging

Untuk log yang lebih detail, gunakan modul logging di pustaka standar, atau pustaka logging pihak ketiga mana pun yang menulis ke `stdout` ataustderr.

Untuk runtime Python yang didukung, Anda dapat memilih apakah log yang dibuat menggunakan `logging` modul standar ditangkap dalam teks biasa atau JSON. Untuk mempelajari selengkapnya, lihat the section called “Menggunakan kontrol logging lanjutan Lambda dengan Python”.

Saat ini, format log default untuk semua runtime Python adalah teks biasa. Contoh berikut menunjukkan bagaimana output log yang dibuat menggunakan `logging` modul standar ditangkap dalam teks biasa di CloudWatch Log.

```
import os
import logging
logger = logging.getLogger()
logger.setLevel("INFO")

def lambda_handler(event, context):
    logger.info('## ENVIRONMENT VARIABLES')
    logger.info(os.environ['AWS_LAMBDA_LOG_GROUP_NAME'])
    logger.info(os.environ['AWS_LAMBDA_LOG_STREAM_NAME'])
    logger.info('## EVENT')
    logger.info(event)
```

Output dari `logger` mencakup tingkat log, stempel waktu, dan ID permintaan.

```
START RequestId: 1c8df7d3-xmpl-46da-9778-518e6eca8125 Version: $LATEST
[INFO]  2025-08-31T22:12:58.534Z    1c8df7d3-xmpl-46da-9778-518e6eca8125    ##
 ENVIRONMENT VARIABLES
[INFO]  2025-08-31T22:12:58.534Z    1c8df7d3-xmpl-46da-9778-518e6eca8125    /aws/
lambda/my-function
[INFO]  2025-08-31T22:12:58.534Z    1c8df7d3-xmpl-46da-9778-518e6eca8125    2025/01/31/
[$LATEST]1bbe51xmplb34a2788dbaa7433b0aa4d
[INFO]  2025-08-31T22:12:58.535Z    1c8df7d3-xmpl-46da-9778-518e6eca8125    ## EVENT
[INFO]  2025-08-31T22:12:58.535Z    1c8df7d3-xmpl-46da-9778-518e6eca8125    {'key':
 'value'}
END RequestId: 1c8df7d3-xmpl-46da-9778-518e6eca8125
REPORT RequestId: 1c8df7d3-xmpl-46da-9778-518e6eca8125  Duration: 2.75 ms   Billed
 Duration: 117 ms Memory Size: 128 MB Max Memory Used: 56 MB  Init Duration: 113.51 ms
XRAY TraceId: 1-5e34a66a-474xmpl7c2534a87870b4370   SegmentId: 073cxmpl3e442861
 Sampled: true
```

Note

Ketika format log fungsi Anda disetel ke teks biasa, pengaturan tingkat log default untuk runtime Python adalah WARN. Ini berarti bahwa Lambda hanya mengirimkan output log dari level WARN dan lebih rendah ke Log. CloudWatch Untuk mengubah tingkat log default, gunakan `logging` `setLevel()` metode Python seperti yang ditunjukkan dalam kode contoh ini. Jika Anda menyetel format log fungsi Anda ke JSON, sebaiknya Anda mengonfigurasi level log fungsi Anda menggunakan Lambda Advanced Logging Controls dan bukan dengan

menyetel level log dalam kode. Untuk mempelajari selengkapnya, lihat the section called "Menggunakan penyaringan tingkat log dengan Python"

Menggunakan kontrol logging lanjutan Lambda dengan Python

Untuk memberi Anda kontrol lebih besar atas bagaimana log fungsi Anda ditangkap, diproses, dan dikonsumsi, Anda dapat mengonfigurasi opsi logging berikut untuk runtime Lambda Python yang didukung:

- Format log - pilih antara teks biasa dan format JSON terstruktur untuk log fungsi Anda
- Tingkat log - untuk log dalam format JSON, pilih tingkat detail log yang dikirim Lambda ke CloudWatch Amazon, seperti ERROR, DEBUG, atau INFO
- Grup log - pilih grup CloudWatch log yang dikirimkan oleh fungsi Anda

Untuk informasi selengkapnya tentang opsi pencatatan ini, dan petunjuk tentang cara mengonfigurasi fungsi Anda untuk menggunakannya, lihatthe section called "Mengkonfigurasi kontrol logging lanjutan untuk Lambda Event Functions".

Untuk mempelajari lebih lanjut tentang menggunakan format log dan opsi tingkat log dengan fungsi Python Anda, lihat panduan di bagian berikut.

Menggunakan log JSON terstruktur dengan Python

Jika Anda memilih JSON untuk format log fungsi Anda, Lambda mengirimkan keluaran log oleh pustaka CloudWatch logging standar Python ke JSON terstruktur. Setiap objek log JSON berisi setidaknya empat pasangan nilai kunci dengan kunci berikut:

- `"timestamp"`- waktu pesan log dihasilkan
- `"level"`- tingkat log yang ditetapkan untuk pesan
- `"message"`- isi pesan log
- `"requestId"`- ID permintaan unik untuk pemanggilan fungsi

`logging`Pustaka Python juga dapat menambahkan pasangan nilai kunci tambahan seperti `"logger"` ke objek JSON ini.

Contoh di bagian berikut menunjukkan bagaimana output log yang dihasilkan menggunakan pustaka `logging` Python ditangkap CloudWatch di Log saat Anda mengonfigurasi format log fungsi Anda sebagai JSON.

Perhatikan bahwa jika Anda menggunakan `print` metode untuk menghasilkan output log dasar seperti yang dijelaskan dalamthe section called “Mencetak ke log”, Lambda menangkap output ini sebagai teks biasa, bahkan jika Anda mengonfigurasi format logging fungsi Anda sebagai JSON.

Output log JSON standar menggunakan pustaka logging Python

Contoh cuplikan kode dan keluaran log berikut menunjukkan bagaimana output log standar yang dihasilkan menggunakan `logging` pustaka Python ditangkap di CloudWatch Log saat format log fungsi Anda disetel ke JSON.

Example Kode pencatatan Python

```
import logging
logger = logging.getLogger()

def lambda_handler(event, context):
    logger.info("Inside the handler function")
```

Example Catatan log JSON

```
{
    "timestamp":"2025-10-27T19:17:45.586Z",
    "level":"INFO",
    "message":"Inside the handler function",
    "logger": "root",
    "requestId":"79b4f56e-95b1-4643-9700-2807f4e68189"
}
```

Mencatat parameter tambahan di JSON

Ketika format log fungsi Anda diatur ke JSON, Anda juga dapat mencatat parameter tambahan dengan pustaka `logging` Python standar dengan menggunakan kata kunci untuk `extra` meneruskan kamus Python ke output log.

Example Kode pencatatan Python

```
import logging
```

```
def lambda_handler(event, context):
    logging.info(
        "extra parameters example",
        extra={"a":"b", "b": [3]},
    )
```

Example Catatan log JSON

```
{
  "timestamp": "2025-11-02T15:26:28Z",
  "level": "INFO",
  "message": "extra parameters example",
  "logger": "root",
  "requestId": "3dbd5759-65f6-45f8-8d7d-5bdc79a3bd01",
  "a": "b",
  "b": [
    3
  ]
}
```

Pengecualian logging di JSON

Cuplikan kode berikut menunjukkan bagaimana pengecualian Python ditangkap dalam output log fungsi Anda saat Anda mengonfigurasi format log sebagai JSON. Perhatikan bahwa output log yang dihasilkan menggunakan `logging.exception` ditetapkan ERROR tingkat log.

Example Kode pencatatan Python

```
import logging

def lambda_handler(event, context):
    try:
        raise Exception("exception")
    except:
        logging.exception("msg")
```

Example Catatan log JSON

```
{
  "timestamp": "2025-11-02T16:18:57Z",
  "level": "ERROR",
```

```
  "message": "msg",
  "logger": "root",
  "stackTrace": [
    "  File \"/var/task/lambda_function.py\", line 15, in lambda_handler\n    raise
 Exception(\"exception\")\n"
  ],
  "errorType": "Exception",
  "errorMessage": "exception",
  "requestId": "3f9d155c-0f09-46b7-bdf1-e91dab220855",
  "location": "/var/task/lambda_function.py:lambda_handler:17"
}
```

Log terstruktur JSON dengan alat logging lainnya

Jika kode Anda sudah menggunakan pustaka logging lain, seperti Powertools for AWS Lambda, untuk menghasilkan log terstruktur JSON, Anda tidak perlu membuat perubahan apa pun. AWS Lambda tidak menyandikan dua kali log apa pun yang sudah dikodekan JSON. Bahkan jika Anda mengonfigurasi fungsi Anda untuk menggunakan format log JSON, output logging Anda muncul CloudWatch dalam struktur JSON yang Anda tentukan.

Contoh berikut menunjukkan bagaimana output log yang dihasilkan menggunakan Powertools untuk AWS Lambda paket ditangkap di CloudWatch Log. Format output log ini sama apakah konfigurasi logging fungsi Anda diatur ke JSON atau TEXT. Untuk informasi selengkapnya tentang penggunaan Powertools untuk AWS Lambda, lihat the section called “Menggunakan Powertools untuk AWS Lambda (Python) dan AWS SAM untuk logging terstruktur” dan the section called “Menggunakan Powertools untuk AWS Lambda (Python) dan AWS CDK untuk logging terstruktur”

Example Cuplikan kode logging Python (menggunakan Powertools untuk AWS Lambda)

```
from aws_lambda_powertools import Logger

logger = Logger()

def lambda_handler(event, context):
    logger.info("Inside the handler function")
```

Example Catatan log JSON (menggunakan Powertools untuk AWS Lambda)

```
{
    "level": "INFO",
    "location": "lambda_handler:7",
```

```
    "message": "Inside the handler function",
    "timestamp": "2025-10-31 22:38:21,010+0000",
    "service": "service_undefined",
    "xray_trace_id": "1-654181dc-65c15d6b0fecbdd1531ecb30"
}
```

Menggunakan penyaringan tingkat log dengan Python

Dengan mengonfigurasi penyaringan tingkat log, Anda dapat memilih untuk mengirim hanya log dari tingkat logging tertentu atau lebih rendah ke Log. CloudWatch Untuk mempelajari cara mengonfigurasi pemfilteran tingkat log untuk fungsi Anda, lihat. the section called “Log-level penyaringan”

AWS Lambda Untuk memfilter log aplikasi Anda sesuai dengan tingkat lognya, fungsi Anda harus menggunakan log berformat JSON. Anda dapat mencapai ini dengan dua cara:

- Buat output log menggunakan pustaka `logging` Python standar dan konfigurasikan fungsi Anda untuk menggunakan pemformatan log JSON. AWS Lambda kemudian memfilter output log Anda menggunakan pasangan nilai kunci “level” di objek JSON yang dijelaskan dalam. the section called “Menggunakan log JSON terstruktur dengan Python” Untuk mempelajari cara mengonfigurasi format log fungsi Anda, lihatthe section called “Mengkonfigurasi kontrol logging lanjutan untuk Lambda Event Functions”.
- Gunakan pustaka atau metode logging lain untuk membuat log terstruktur JSON dalam kode Anda yang menyertakan pasangan nilai kunci “level” yang menentukan tingkat keluaran log. Misalnya, Anda dapat menggunakan Powertools AWS Lambda untuk menghasilkan output log terstruktur JSON dari kode Anda.

  Anda juga dapat menggunakan pernyataan cetak untuk menampilkan objek JSON yang berisi pengenal tingkat log. Pernyataan cetak berikut menghasilkan output diformat JSON di mana tingkat log diatur ke INFO. AWS Lambda mengirimkan objek JSON ke CloudWatch Log jika level logging fungsi Anda disetel ke INFO, DEBUG, atau TRACE.

  ```
  print('{"msg":"My log message", "level":"info"}')
  ```

Agar Lambda dapat memfilter log fungsi Anda, Anda juga harus menyertakan pasangan nilai `"timestamp"` kunci dalam keluaran log JSON Anda. Waktu harus ditentukan dalam format stempel waktu RFC 3339 yang valid. Jika Anda tidak menyediakan stempel waktu yang valid, Lambda menetapkan log INFO level dan menambahkan stempel waktu untuk Anda.

Melihat log di konsol Lambda

Anda dapat menggunakan konsol Lambda untuk melihat keluaran log setelah Anda menjalankan Fungsi Peristiwa Lambda.

Jika kode Anda dapat diuji dari editor Kode tertanam, Anda menemukan log dalam hasil eksekusi. Saat Anda menggunakan fitur pengujian konsol untuk menjalankan fungsi, Anda menemukan Keluaran Log di bagian Detail.

Melihat log di CloudWatch konsol

Anda dapat menggunakan CloudWatch konsol Amazon untuk melihat log untuk semua pemanggilan Fungsi Peristiwa Lambda.

Untuk melihat log di CloudWatch konsol

1. Buka halaman Grup log di CloudWatch konsol.
2. Pilih grup log untuk fungsi Anda (/aws/lambda/***your-function-name***).
3. Pilih pengaliran log.

Setiap aliran log sesuai dengan instans fungsi Anda. Aliran log muncul saat Anda memperbarui fungsi Lambda, dan saat instance tambahan dibuat untuk menangani pemanggilan bersamaan. Untuk menemukan log untuk pemanggilan tertentu, kami sarankan untuk menginstrumentasi fungsi Anda dengan. AWS X-Ray X-Ray mencatat detail tentang permintaan dan aliran log dalam jejak.

Melihat log dengan AWS CLI

AWS CLI Ini adalah alat sumber terbuka yang dapat Anda gunakan untuk berinteraksi dengan AWS layanan menggunakan perintah di shell baris perintah Anda. Untuk menyelesaikan langkah-langkah di bagian ini, Anda harus memiliki AWS CLI versi 2.

Anda dapat menggunakan AWS CLI untuk mengambil log untuk invokasi menggunakan opsi perintah `--log-type`. Respons berisi bidang `LogResult` yang memuat hingga 4 KB log berkode base64 dari invokasi.

Example mengambil ID log

Contoh berikut menunjukkan cara mengambil ID log dari `LogResult` untuk fungsi bernama `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

Anda akan melihat output berikut:

```
{
    "StatusCode": 200,
    "LogResult":
 "U1RBUlQgUmVxdWVzdElkOiA4N2QwNDRiOC1mMTU0LTExZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "ExecutedVersion": "$LATEST"
}
```

Example mendekode log

Pada prompt perintah yang sama, gunakan utilitas `base64` untuk mendekodekan log. Contoh berikut menunjukkan cara mengambil log berkode base64 untuk `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail \
--query 'LogResult' --output text --cli-binary-format raw-in-base64-out | base64 --
decode
```

cli-binary-formatOpsi ini diperlukan jika Anda menggunakan AWS CLI versi 2. Untuk menjadikan ini pengaturan default, jalankan`aws configure set cli-binary-format raw-in-base64-out`. Untuk informasi selengkapnya, lihat opsi baris perintah global yang AWS CLI didukung di Panduan AWS Command Line Interface Pengguna untuk Versi 2.

Anda akan melihat output berikut:

```
START RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Version: $LATEST
"AWS_SESSION_TOKEN": "AgoJb3JpZ2luX2VjELj...", "_X_AMZN_TRACE_ID": "Root=1-5d02e5ca-
f5792818b6fe8368e5b51d50;Parent=191db58857df8395;Sampled=0"",ask/lib:/opt/lib",
END RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8
REPORT RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8  Duration: 79.67 ms      Billed
 Duration: 80 ms         Memory Size: 128 MB     Max Memory Used: 73 MB
```

Utilitas `base64` tersedia di Linux, macOS, dan Ubuntu pada Windows. Pengguna macOS mungkin harus menggunakan `base64 -D`.

Example Skrip get-logs.sh

Pada prompt perintah yang sama, gunakan script berikut untuk mengunduh lima peristiwa log terakhir. Skrip menggunakan `sed` untuk menghapus kutipan dari file output, dan akan tidur selama 15

detik untuk memberikan waktu agar log tersedia. Output mencakup respons dari Lambda dan output dari perintah `get-log-events`.

Salin konten dari contoh kode berikut dan simpan dalam direktori proyek Lambda Anda sebagai `get-logs.sh`.

cli-binary-formatOpsi ini diperlukan jika Anda menggunakan AWS CLI versi 2. Untuk menjadikan ini pengaturan default, jalankan`aws configure set cli-binary-format raw-in-base64-out`. Untuk informasi selengkapnya, lihat opsi baris perintah global yang AWS CLI didukung di Panduan AWS Command Line Interface Pengguna untuk Versi 2.

```
#!/bin/bash
aws lambda invoke --function-name my-function --cli-binary-format raw-in-base64-out --
payload '{"key": "value"}' out
sed -i'' -e 's/"//g' out
sleep 15
aws logs get-log-events --log-group-name /aws/lambda/my-function --log-stream-
name stream1 --limit 5
```

Example macOS dan Linux (khusus)

Dalam prompt perintah yang sama, pengguna macOS dan Linux mungkin perlu menjalankan perintah berikut untuk memastikan skrip dapat dieksekusi.

```
chmod -R 755 get-logs.sh
```

Example mengambil lima log acara terakhir

Pada prompt perintah yang sama, gunakan skrip berikut untuk mendapatkan lima log acara terakhir.

```
./get-logs.sh
```

Anda akan melihat output berikut:

```
{
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}
{
```

```
    "events": [
        {
            "timestamp": 1559763003171,
            "message": "START RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf Version:
 $LATEST\n",
            "ingestionTime": 1559763003309
        },
        {
            "timestamp": 1559763003173,
            "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tENVIRONMENT VARIABLES\r{\r  \"AWS_LAMBDA_FUNCTION_VERSION\": \"$LATEST\",
\r ...",
            "ingestionTime": 1559763018353
        },
        {
            "timestamp": 1559763003173,
            "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tEVENT\r{\r  \"key\": \"value\"\r}\n",
            "ingestionTime": 1559763018353
        },
        {
            "timestamp": 1559763003218,
            "message": "END RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf\n",
            "ingestionTime": 1559763018353
        },
        {
            "timestamp": 1559763003218,
            "message": "REPORT RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tDuration: 26.73 ms\tBilled Duration: 27 ms \tMemory Size: 128 MB\tMax Memory Used: 75
 MB\t\n",
            "ingestionTime": 1559763018353
        }
    ],
    "nextForwardToken": "f/34783877304859518393868359594929986069206639495374241795",
    "nextBackwardToken": "b/34783877303811383369537420289090800615709599058929582080"
}
```


Menghapus log

Grup log tidak terhapus secara otomatis ketika Anda menghapus suatu fungsi. Untuk menghindari penyimpanan log secara tidak terbatas, hapus kelompok log, atau lakukan konfigurasi periode penyimpanan, yang setelahnya log akan dihapus secara otomatis.

Menggunakan alat logging dan pustaka lainnya

Powertools for AWS Lambda (Python) adalah toolkit pengembang untuk mengimplementasikan praktik terbaik Tanpa Server dan meningkatkan kecepatan pengembang. Utilitas Logger menyediakan logger yang dioptimalkan Lambda yang mencakup informasi tambahan tentang konteks fungsi di semua fungsi Anda dengan output terstruktur sebagai JSON. Gunakan utilitas ini untuk melakukan hal berikut:

- Tangkap bidang kunci dari konteks Lambda, cold start, dan struktur logging output sebagai JSON
- Log peristiwa pemanggilan Lambda saat diinstruksikan (dinonaktifkan secara default)
- Cetak semua log hanya untuk persentase pemanggilan melalui pengambilan sampel log (dinonaktifkan secara default)
- Tambahkan kunci tambahan ke log terstruktur kapan saja
- Gunakan pemformat log kustom (Bring Your Own Formatter) untuk mengeluarkan log dalam struktur yang kompatibel dengan Logging RFC organisasi Anda

Menggunakan Powertools untuk AWS Lambda (Python) dan AWS SAM untuk logging terstruktur

Ikuti langkah-langkah di bawah ini untuk mengunduh, membangun, dan menyebarkan contoh aplikasi Hello World Python dengan Powertools terintegrasi untuk modul Python menggunakan aplikasi. AWS SAM Aplikasi ini mengimplementasikan backend API dasar dan menggunakan Powertools untuk memancarkan log, metrik, dan jejak. Ini terdiri dari titik akhir Amazon API Gateway dan Fungsi Peristiwa Lambda. Saat Anda mengirim permintaan GET ke titik akhir API Gateway, fungsi tersebut memanggil, mengirim log dan metrik menggunakan Format Metrik Tertanam ke CloudWatch, dan mengirimkan jejak ke. AWS X-Ray Fungsi mengembalikan `hello world` pesan.

Prasyarat

Untuk menyelesaikan langkah-langkah di bagian ini, Anda harus memiliki hal-hal berikut:

- Python 3.9
- AWS CLI versi 2
- AWS SAM CLI versi 1.75 atau yang lebih baru. Jika Anda memiliki versi CLI yang lebih lama, lihat Memutakhirkan AWS SAM CLI. AWS SAM

Menyebarkan sampel AWS SAM application

1. Inisialisasi aplikasi menggunakan template Hello World Python.

```
sam init --app-template hello-world-powertools-python --name sam-app --package-type
 Zip --runtime python3.9 --no-tracing
```

2. Bangun aplikasi.

```
cd sam-app && sam build
```

3. Terapkan aplikasi.

```
sam deploy --guided
```

4. Ikuti petunjuk di layar. Untuk menerima opsi default yang disediakan dalam pengalaman interaktif, tekan`Enter`.

> **Note**
>
> Karena HelloWorldFunction mungkin tidak memiliki otorisasi yang ditentukan, Apakah ini baik-baik saja? , pastikan untuk masuky.

5. Dapatkan URL aplikasi yang digunakan:

```
aws cloudformation describe-stacks --stack-name sam-app --query
 'Stacks[0].Outputs[?OutputKey==`HelloWorldApi`].OutputValue' --output text
```

6. Memanggil titik akhir API:

```
curl GET <URL_FROM_PREVIOUS_STEP>
```

Jika berhasil, Anda akan melihat tanggapan ini:

```
{"message":"hello world"}
```

7. Untuk mendapatkan log untuk fungsi tersebut, jalankan log sam. Untuk informasi selengkapnya, lihat Bekerja dengan log di Panduan AWS Serverless Application Model Pengembang.

```
sam logs --stack-name sam-app
```

Output log terlihat seperti ini:

```
2025/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04
 2025-02-03T14:59:50.371000 INIT_START Runtime Version:
 python:3.9.v16    Runtime Version ARN: arn:aws:lambda:us-
east-1::runtime:07a48df201798d627f2b950f03bb227aab4a655a1d019c3296406f95937e2525
2025/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04 2025-02-03T14:59:51.112000
 START RequestId: d455cfc4-7704-46df-901b-2a5cce9405be Version: $LATEST
2025/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04 2025-02-03T14:59:51.114000 {
  "level": "INFO",
  "location": "hello:23",
  "message": "Hello world API - HTTP 200",
  "timestamp": "2025-02-03 14:59:51,113+0000",
  "service": "PowertoolsHelloWorld",
  "cold_start": true,
  "function_name": "sam-app-HelloWorldFunction-YBg8yfYtOc9j",
  "function_memory_size": "128",
  "function_arn": "arn:aws:lambda:us-east-1:111122223333:function:sam-app-
HelloWorldFunction-YBg8yfYtOc9j",
  "function_request_id": "d455cfc4-7704-46df-901b-2a5cce9405be",
  "correlation_id": "e73f8aef-5e07-436e-a30b-63e4b23f0047",
  "xray_trace_id": "1-63dd2166-434a12c22e1307ff2114f299"
}
2025/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04 2025-02-03T14:59:51.126000 {
  "_aws": {
    "Timestamp": 1675436391126,
    "CloudWatchMetrics": [
      {
        "Namespace": "Powertools",
        "Dimensions": [
          [
            "function_name",
            "service"
          ]
        ],
        "Metrics": [
          {
            "Name": "ColdStart",
            "Unit": "Count"
          }
        ]
      }
    ]
```

```
  },
  "function_name": "sam-app-HelloWorldFunction-YBg8yfYtOc9j",
  "service": "PowertoolsHelloWorld",
  "ColdStart": [
    1.0
  ]
}
2025/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04 2025-02-03T14:59:51.126000 {
  "_aws": {
    "Timestamp": 1675436391126,
    "CloudWatchMetrics": [
      {
        "Namespace": "Powertools",
        "Dimensions": [
          [
            "service"
          ]
        ],
        "Metrics": [
          {
            "Name": "HelloWorldInvocations",
            "Unit": "Count"
          }
        ]
      }
    ]
  },
  "service": "PowertoolsHelloWorld",
  "HelloWorldInvocations": [
    1.0
  ]
}
2025/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04 2025-02-03T14:59:51.128000 END
 RequestId: d455cfc4-7704-46df-901b-2a5cce9405be
2025/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04 2025-02-03T14:59:51.128000
 REPORT RequestId: d455cfc4-7704-46df-901b-2a5cce9405be    Duration: 16.33 ms
 Billed Duration: 756 ms    Memory Size: 128 MB    Max Memory Used: 64 MB    Init
 Duration: 739.46 ms
XRAY TraceId: 1-63dd2166-434a12c22e1307ff2114f299    SegmentId: 3c5d18d735a1ced0
 Sampled: true
```

8. Ini adalah titik akhir API publik yang dapat diakses melalui internet. Kami menyarankan Anda menghapus titik akhir setelah pengujian.

```
sam delete
```

Mengelola retensi log

Grup log tidak terhapus secara otomatis ketika Anda menghapus suatu fungsi. Untuk menghindari penyimpanan log tanpa batas waktu, hapus grup log, atau konfigurasikan periode retensi setelah itu secara CloudWatch otomatis menghapus log. Untuk mengatur penyimpanan log, tambahkan yang berikut ini ke AWS SAM templat Anda:

```
Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function
    Properties:
      # Omitting other properties

  LogGroup:
    Type: AWS::Logs::LogGroup
    Properties:
      LogGroupName: !Sub "/aws/lambda/${HelloWorldFunction}"
      RetentionInDays: 7
```

Menggunakan Powertools untuk AWS Lambda (Python) dan AWS CDK untuk logging terstruktur

Ikuti langkah-langkah di bawah ini untuk mengunduh, membangun, dan menyebarkan contoh aplikasi Hello World Python dengan Powertools terintegrasi untuk modul AWS Lambda (Python) menggunakan modul. AWS CDK Aplikasi ini mengimplementasikan backend API dasar dan menggunakan Powertools untuk memancarkan log, metrik, dan jejak. Ini terdiri dari titik akhir Amazon API Gateway dan Fungsi Peristiwa Lambda. Saat Anda mengirim permintaan GET ke titik akhir API Gateway, fungsi tersebut memanggil, mengirim log dan metrik menggunakan Format Metrik Tertanam ke CloudWatch, dan mengirimkan jejak ke. AWS X-Ray Fungsi mengembalikan pesan hello world.

Prasyarat

Untuk menyelesaikan langkah-langkah di bagian ini, Anda harus memiliki hal-hal berikut:

- Python 3.9
- AWS CLI versi 2
- AWS CDK versi 2

- AWS SAM CLI versi 1.75 atau yang lebih baru. Jika Anda memiliki versi CLI yang lebih lama, lihat Memutakhirkan AWS SAM CLI. AWS SAM

Menyebarkan sampel AWS CDK application

1. Buat direktori proyek untuk aplikasi baru Anda.

```
mkdir hello-world
cd hello-world
```

2. Inisialisasi aplikasi.

```
cdk init app --language python
```

3. Instal dependensi Python.

```
pip install -r requirements.txt
```

4. Buat direktori lambda_function di bawah folder root.

```
mkdir lambda_function
cd lambda_function
```

5. Buat file app.py dan tambahkan kode berikut ke file. Ini adalah kode untuk fungsi tersebut.

```
from aws_lambda_powertools.event_handler import APIGatewayRestResolver
from aws_lambda_powertools.utilities.typing import LambdaContext
from aws_lambda_powertools.logging import correlation_paths
from aws_lambda_powertools import Logger
from aws_lambda_powertools import Tracer
from aws_lambda_powertools import Metrics
from aws_lambda_powertools.metrics import MetricUnit

app = APIGatewayRestResolver()
tracer = Tracer()
logger = Logger()
metrics = Metrics(namespace="PowertoolsSample")

@app.get("/hello")
@tracer.capture_method
def hello():
    # adding custom metrics
```

```
    # See: https://docs.aws.amazon.com/powertools/python/latest//latest/core/
metrics/
    metrics.add_metric(name="HelloWorldInvocations", unit=MetricUnit.Count,
 value=1)

    # structured log
    # See: https://docs.aws.amazon.com/powertools/python/latest//latest/core/
logger/
    logger.info("Hello world API - HTTP 200")
    return {"message": "hello world"}

# Enrich logging with contextual information from Lambda
@logger.inject_lambda_context(correlation_id_path=correlation_paths.API_GATEWAY_REST)
# Adding tracer
# See: https://docs.aws.amazon.com/powertools/python/latest//latest/core/tracer/
@tracer.capture_lambda_handler
# ensures metrics are flushed upon request completion/failure and capturing
 ColdStart metric
@metrics.log_metrics(capture_cold_start_metric=True)
def lambda_handler(event: dict, context: LambdaContext) -> dict:
    return app.resolve(event, context)
```

6. Buka direktori hello_world. Anda akan melihat file bernama hello_world_stack.py.

```
cd ..
cd hello_world
```

7. Buka hello_world_stack.py dan tambahkan kode berikut ke file. Ini berisi Konstruktor Lambda, yang membuat fungsi, mengonfigurasi variabel lingkungan untuk Powertools dan menetapkan retensi log ke satu minggu, dan Konstruktor ApiGatewayv 1, yang membuat REST API.

```
from aws_cdk import (
    Stack,
    aws_apigateway as apigwv1,
    aws_lambda as lambda_,
    CfnOutput,
    Duration
)
from constructs import Construct

class HelloWorldStack(Stack):

    def __init__(self, scope: Construct, construct_id: str, **kwargs) -> None:
```

```
        super().__init__(scope, construct_id, **kwargs)

        # Powertools Lambda Layer
        powertools_layer = lambda_.LayerVersion.from_layer_version_arn(
            self,
            id="lambda-powertools",
            # Using AWS Lambda Powertools with Lambda Layer
            # This imports the Powertools layer which provides observability
 features for Lambda functions
            # For available versions, see: https://docs.aws.amazon.com/powertools/
python/latest/#lambda-layer
        )

        function = lambda_.Function(self,
            'sample-app-lambda',
            runtime=lambda_.Runtime.PYTHON_3_9,
            layers=[powertools_layer],
            code = lambda_.Code.from_asset("./lambda_function/"),
            handler="app.lambda_handler",
            memory_size=128,
            timeout=Duration.seconds(3),
            architecture=lambda_.Architecture.X86_64,
            environment={
                "POWERTOOLS_SERVICE_NAME": "PowertoolsHelloWorld",
                "POWERTOOLS_METRICS_NAMESPACE": "PowertoolsSample",
                "LOG_LEVEL": "INFO"
            }
        )

        apigw = apigwv1.RestApi(self, "PowertoolsAPI",
 deploy_options=apigwv1.StageOptions(stage_name="dev"))

        hello_api = apigw.root.add_resource("hello")
        hello_api.add_method("GET", apigwv1.LambdaIntegration(function,
 proxy=True))

        CfnOutput(self, "apiUrl", value=f"{apigw.url}hello")
```

8. Men-deploy aplikasi Anda.

```
cd ..
cdk deploy
```

9. Dapatkan URL aplikasi yang digunakan:

```
aws cloudformation describe-stacks --stack-name HelloWorldStack --query
 'Stacks[0].Outputs[?OutputKey==`apiUrl`].OutputValue' --output text
```

10. Memanggil titik akhir API:

```
curl GET <URL_FROM_PREVIOUS_STEP>
```

Jika berhasil, Anda akan melihat tanggapan ini:

```
{"message":"hello world"}
```

11. Untuk mendapatkan log untuk fungsi tersebut, jalankan log sam. Untuk informasi selengkapnya, lihat Bekerja dengan log di Panduan AWS Serverless Application Model Pengembang.

```
sam logs --stack-name HelloWorldStack
```

Output log terlihat seperti ini:

```
2025/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04
 2025-02-03T14:59:50.371000 INIT_START Runtime Version:
 python:3.9.v16    Runtime Version ARN: arn:aws:lambda:us-
east-1::runtime:07a48df201798d627f2b950f03bb227aab4a655a1d019c3296406f95937e2525
2025/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04 2025-02-03T14:59:51.112000
 START RequestId: d455cfc4-7704-46df-901b-2a5cce9405be Version: $LATEST
2025/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04 2025-02-03T14:59:51.114000 {
  "level": "INFO",
  "location": "hello:23",
    "message": "Hello world API - HTTP 200",
  "timestamp": "2025-02-03 14:59:51,113+0000",
  "service": "PowertoolsHelloWorld",
  "cold_start": true,
  "function_name": "sam-app-HelloWorldFunction-YBg8yfYtOc9j",
  "function_memory_size": "128",
  "function_arn": "arn:aws:lambda:us-east-1:111122223333:function:sam-app-
HelloWorldFunction-YBg8yfYtOc9j",
  "function_request_id": "d455cfc4-7704-46df-901b-2a5cce9405be",
  "correlation_id": "e73f8aef-5e07-436e-a30b-63e4b23f0047",
  "xray_trace_id": "1-63dd2166-434a12c22e1307ff2114f299"
}
2025/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04 2025-02-03T14:59:51.126000 {
```

```
  "_aws": {
    "Timestamp": 1675436391126,
    "CloudWatchMetrics": [
      {
        "Namespace": "Powertools",
        "Dimensions": [
          [
            "function_name",
            "service"
          ]
        ],
        "Metrics": [
          {
            "Name": "ColdStart",
            "Unit": "Count"
          }
        ]
      }
    ]
  },
  "function_name": "sam-app-HelloWorldFunction-YBg8yfYtOc9j",
  "service": "PowertoolsHelloWorld",
  "ColdStart": [
    1.0
  ]
}
2025/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04 2025-02-03T14:59:51.126000 {
  "_aws": {
    "Timestamp": 1675436391126,
    "CloudWatchMetrics": [
      {
        "Namespace": "Powertools",
        "Dimensions": [
          [
            "service"
          ]
        ],
        "Metrics": [
          {
            "Name": "HelloWorldInvocations",
            "Unit": "Count"
          }
        ]
      }
```

```
    ]
  },
  "service": "PowertoolsHelloWorld",
  "HelloWorldInvocations": [
    1.0
  ]
}
2025/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04 2025-02-03T14:59:51.128000 END
 RequestId: d455cfc4-7704-46df-901b-2a5cce9405be
2025/02/03/[$LATEST]ea9a64ec87294bf6bbc9026c05a01e04 2025-02-03T14:59:51.128000
 REPORT RequestId: d455cfc4-7704-46df-901b-2a5cce9405be    Duration: 16.33 ms
 Billed Duration: 756 ms    Memory Size: 128 MB    Max Memory Used: 64 MB    Init
 Duration: 739.46 ms
XRAY TraceId: 1-63dd2166-434a12c22e1307ff2114f299    SegmentId: 3c5d18d735a1ced0
 Sampled: true
```

12. Ini adalah titik akhir API publik yang dapat diakses melalui internet. Kami menyarankan Anda menghapus titik akhir setelah pengujian.

```
cdk destroy
```

AWS Lambda Pengujian Fungsi Acara dengan Python

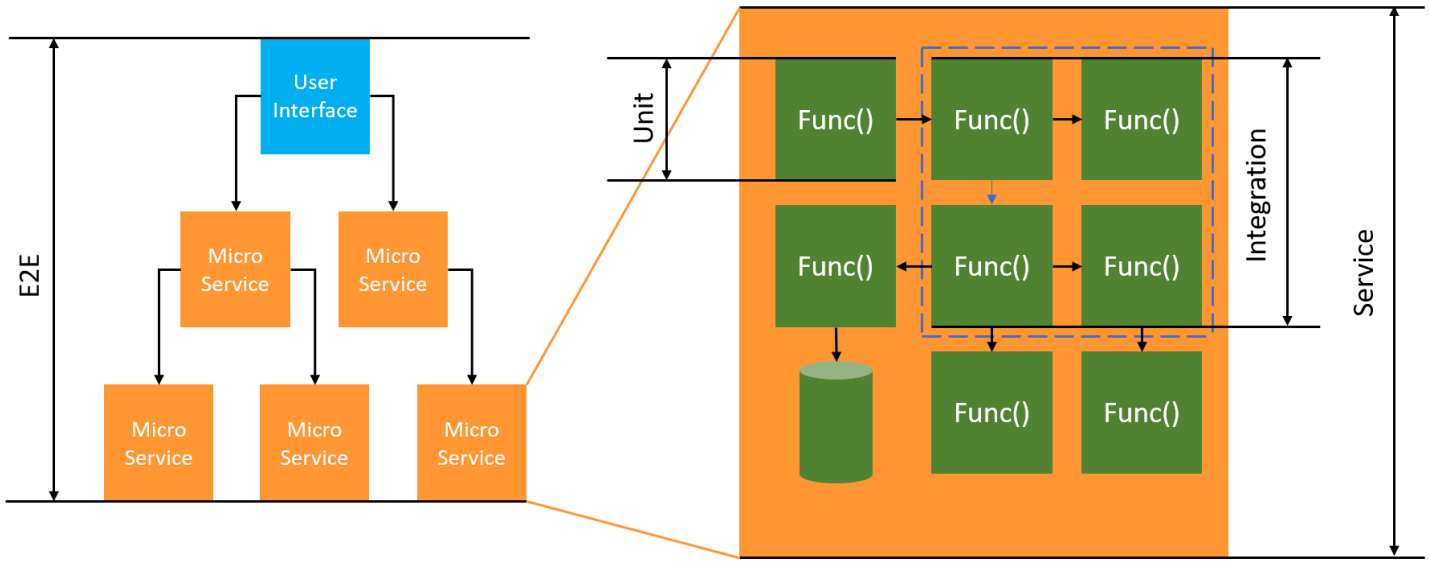
> **Note**
>
> Lihat bagian Fungsi pengujian untuk pengenalan lengkap tentang teknik dan praktik terbaik untuk menguji solusi tanpa server.

Menguji fungsi tanpa server menggunakan jenis dan teknik pengujian tradisional, tetapi Anda juga harus mempertimbangkan pengujian aplikasi tanpa server secara keseluruhan. Cloud-based tes memberikan ukuran kualitas yang paling akurat dari fungsi dan aplikasi tanpa server Anda.

Arsitektur aplikasi tanpa server mencakup layanan terkelola yang menyediakan fungsionalitas aplikasi penting melalui panggilan API. Untuk alasan ini, siklus pengembangan Anda harus menyertakan pengujian otomatis yang memverifikasi fungsionalitas saat fungsi dan layanan Anda berinteraksi.

Jika Anda tidak membuat pengujian berbasis cloud, Anda dapat mengalami masalah karena perbedaan antara lingkungan lokal dan lingkungan yang diterapkan. Proses integrasi berkelanjutan Anda harus menjalankan pengujian terhadap serangkaian sumber daya yang disediakan di cloud sebelum mempromosikan kode Anda ke lingkungan penerapan berikutnya, seperti QA, Staging, atau Production.

Lanjutkan membaca panduan singkat ini untuk mempelajari strategi pengujian untuk aplikasi tanpa server, atau kunjungi repositori Sampel Uji Tanpa Server untuk menyelami contoh-contoh praktis, khusus untuk bahasa dan runtime pilihan Anda.



Untuk pengujian tanpa server, Anda masih menulis pengujian unit, integrasi, dan ujung-ke-ujung.

- Tes unit - Tes yang dijalankan terhadap blok kode yang terisolasi. Misalnya, memverifikasi logika bisnis untuk menghitung biaya pengiriman yang diberikan item dan tujuan tertentu.
- Tes integrasi - Tes yang melibatkan dua atau lebih komponen atau layanan yang berinteraksi, biasanya di lingkungan cloud. Misalnya, memverifikasi fungsi memproses peristiwa dari antrian.
- End-to-end tes - Tes yang memverifikasi perilaku di seluruh aplikasi. Misalnya, memastikan infrastruktur diatur dengan benar dan bahwa peristiwa mengalir antar layanan seperti yang diharapkan untuk merekam pesanan pelanggan.

Menguji aplikasi tanpa server

Anda biasanya menggunakan campuran pendekatan untuk menguji kode aplikasi tanpa server Anda, termasuk pengujian di cloud, pengujian dengan tiruan, dan kadang-kadang menguji dengan emulator.

Pengujian di cloud

Pengujian di cloud sangat berharga untuk semua fase pengujian, termasuk pengujian unit, pengujian integrasi, dan pengujian ujung ke ujung. Anda menjalankan pengujian terhadap kode yang diterapkan di cloud dan berinteraksi dengan layanan berbasis cloud. Pendekatan ini memberikan ukuran kualitas kode Anda yang paling akurat.

Cara mudah untuk men-debug Fungsi Acara Lambda Anda di cloud adalah melalui konsol dengan acara pengujian. Peristiwa pengujian adalah input JSON ke fungsi Anda. Jika fungsi Anda tidak memerlukan input, acara dapat berupa dokumen (`{}`) JSON kosong. Konsol menyediakan

contoh peristiwa untuk berbagai integrasi layanan. Setelah membuat acara di konsol, Anda dapat membagikannya dengan tim Anda untuk membuat pengujian lebih mudah dan konsisten.

> **Note**
>
> Menguji fungsi di konsol adalah cara cepat untuk memulai, tetapi mengotomatiskan siklus pengujian Anda memastikan kualitas aplikasi dan kecepatan pengembangan.

Alat pengujian

Alat dan teknik ada untuk mempercepat loop umpan balik pengembangan. Misalnya, AWS SAM Accelerate dan mode tontonan AWS CDK mengurangi waktu yang diperlukan untuk memperbarui lingkungan cloud.

Moto adalah pustaka Python untuk mengejek AWS layanan dan sumber daya, sehingga Anda dapat menguji fungsi Anda dengan sedikit atau tanpa modifikasi menggunakan dekorator untuk mencegat dan mensimulasikan respons.

Fitur validasi Powertools for (AWS Lambda Python) menyediakan dekorator sehingga Anda dapat memvalidasi peristiwa input dan respons output dari fungsi Python Anda.

Untuk informasi lebih lanjut, baca posting blog Unit Testing Lambda dengan Python dan Mock Services. AWS

Untuk mengurangi latensi yang terkait dengan iterasi penerapan cloud, lihat Mode tontonan Accelerate, AWS Cloud Development Kit (CDK)AWS Serverless Application Model (AWSAWSSAM). Alat-alat ini memantau infrastruktur dan kode Anda untuk perubahan. Mereka bereaksi terhadap perubahan ini dengan membuat dan menerapkan pembaruan tambahan secara otomatis ke lingkungan cloud Anda.

Contoh yang menggunakan alat ini tersedia di repositori kode Sampel Uji Python.

Menginstrumentasi kode Python di AWS Lambda

Lambda terintegrasi AWS X-Ray untuk membantu Anda melacak, men-debug, dan mengoptimalkan aplikasi Lambda. Anda dapat menggunakan X-Ray untuk melacak permintaan saat melintasi sumber daya dalam aplikasi Anda, yang mungkin termasuk fungsi Lambda dan layanan lainnya. AWS

Untuk mengirim data penelusuran X-Ray, Anda dapat menggunakan salah satu dari tiga pustaka SDK:

- AWS Distro for OpenTelemetry (ADOT) — Distribusi SDK (OTel) yang aman, siap produksi, dan AWS didukung. OpenTelemetry
- AWS X-Ray SDK for PythonSDK untuk menghasilkan dan mengirim data jejak ke X-Ray.
- Powertools for AWS Lambda (Python) - Toolkit pengembang untuk mengimplementasikan praktik terbaik Tanpa Server dan meningkatkan kecepatan pengembang.

Setiap SDK menawarkan cara untuk mengirim data telemetri Anda ke layanan. X-Ray Anda kemudian dapat menggunakan X-Ray untuk melihat, memfilter, dan mendapatkan wawasan tentang metrik kinerja aplikasi Anda untuk mengidentifikasi masalah dan peluang untuk pengoptimalan.

> **Important**
>
> The X-Ray dan Powertools untuk AWS Lambda SDK adalah bagian dari solusi instrumentasi terintegrasi ketat yang ditawarkan oleh. AWS Lapisan Lambda ADOT adalah bagian dari standar industri untuk melacak instrumentasi yang mengumpulkan lebih banyak data secara umum, tetapi mungkin tidak cocok untuk semua kasus penggunaan. Anda dapat menerapkan penelusuran ujung ke ujung dalam X-Ray menggunakan salah satu solusi. Untuk mempelajari lebih lanjut tentang memilih di antara keduanya, lihat Memilih antara AWS Distro untuk Telemetri Terbuka dan SDK. X-Ray

Bagian-bagian

- Menggunakan Powertools untuk AWS Lambda (Python) dan AWS SAM untuk menelusuri
- Menggunakan Powertools untuk AWS Lambda (Python) dan AWS CDK untuk menelusuri
- Menggunakan ADOT untuk instrumen fungsi Python Anda
- Menggunakan X-Ray SDK untuk instrumen fungsi Python Anda
- Mengaktifkan penelusuran dengan konsol Lambda

- Mengaktifkan penelusuran dengan Lambda API
- Mengaktifkan penelusuran dengan CloudFormation
- Menafsirkan jejak X-Ray
- Menyimpan dependensi runtime dalam lapisan (SDK) X-Ray

Menggunakan Powertools untuk AWS Lambda (Python) dan AWS SAM untuk menelusuri

Ikuti langkah-langkah di bawah ini untuk mengunduh, membangun, dan menyebarkan contoh aplikasi Hello World Python dengan Powertools terintegrasi untuk modul AWS Lambda (Python) menggunakan modul. AWS SAM Aplikasi ini mengimplementasikan backend API dasar dan menggunakan Powertools untuk memancarkan log, metrik, dan jejak. Ini terdiri dari titik akhir Amazon API Gateway dan Fungsi Peristiwa Lambda. Saat Anda mengirim permintaan GET ke titik akhir API Gateway, fungsi tersebut memanggil, mengirim log dan metrik menggunakan Format Metrik Tertanam ke CloudWatch, dan mengirimkan jejak ke. AWS X-Ray Fungsi mengembalikan pesan hello world.

Prasyarat

Untuk menyelesaikan langkah-langkah di bagian ini, Anda harus memiliki hal-hal berikut:

- Python 3.11
- AWS CLI versi 2
- AWS SAM CLI versi 1.75 atau yang lebih baru. Jika Anda memiliki versi CLI yang lebih lama, lihat Memutakhirkan AWS SAM CLI. AWS SAM

Menyebarkan sampel AWS SAM application

1. Inisialisasi aplikasi menggunakan template Hello World Python.

```
sam init --app-template hello-world-powertools-python --name sam-app --package-type
 Zip --runtime python3.11 --no-tracing
```

2. Bangun aplikasi.

```
cd sam-app && sam build
```

3. Terapkan aplikasi.

```
sam deploy --guided
```

4. Ikuti petunjuk di layar. Untuk menerima opsi default yang disediakan dalam pengalaman interaktif, tekan`Enter`.

> **Note**
>
> Karena HelloWorldFunction mungkin tidak memiliki otorisasi yang ditentukan, Apakah ini baik-baik saja? , pastikan untuk masuky.

5. Dapatkan URL aplikasi yang digunakan:

```
aws cloudformation describe-stacks --stack-name sam-app --query
 'Stacks[0].Outputs[?OutputKey==`HelloWorldApi`].OutputValue' --output text
```

6. Memanggil titik akhir API:

```
curl -X GET <URL_FROM_PREVIOUS_STEP>
```

Jika berhasil, Anda akan melihat tanggapan ini:

```
{"message":"hello world"}
```

7. Untuk mendapatkan jejak untuk fungsi tersebut, jalankan jejak sam.

```
sam traces
```

Output jejak terlihat seperti ini:

```
New XRay Service Graph
  Start time: 2023-02-03 14:59:50+00:00
  End time: 2023-02-03 14:59:50+00:00
  Reference Id: 0 - (Root) AWS::Lambda - sam-app-HelloWorldFunction-YBg8yfYtOc9j -
 Edges: [1]
   Summary_statistics:
     - total requests: 1
     - ok count(2XX): 1
     - error count(4XX): 0
     - fault count(5XX): 0
```

```
    - total response time: 0.924
  Reference Id: 1 - AWS::Lambda::Function - sam-app-HelloWorldFunction-YBg8yfYtOc9j
 - Edges: []
   Summary_statistics:
     - total requests: 1
     - ok count(2XX): 1
     - error count(4XX): 0
     - fault count(5XX): 0
     - total response time: 0.016
  Reference Id: 2 - client - sam-app-HelloWorldFunction-YBg8yfYtOc9j - Edges: [0]
   Summary_statistics:
     - total requests: 0
     - ok count(2XX): 0
     - error count(4XX): 0
     - fault count(5XX): 0
     - total response time: 0

XRay Event [revision 1] at (2023-02-03T14:59:50.204000) with id
 (1-63dd2166-434a12c22e1307ff2114f299) and duration (0.924s)
 - 0.924s - sam-app-HelloWorldFunction-YBg8yfYtOc9j [HTTP: 200]
 - 0.016s - sam-app-HelloWorldFunction-YBg8yfYtOc9j
   - 0.739s - Initialization
   - 0.016s - Invocation
     - 0.013s - ## lambda_handler
       - 0.000s - ## app.hello
   - 0.000s - Overhead
```

8. Ini adalah titik akhir API publik yang dapat diakses melalui internet. Kami menyarankan Anda menghapus titik akhir setelah pengujian.

```
sam delete
```

X-Ray tidak melacak semua permintaan ke aplikasi Anda. X-Ray menerapkan algoritma pengambilan sampel untuk memastikan bahwa penelusuran efisien, sambil tetap memberikan sampel yang representatif dari semua permintaan. Tingkat pengambilan sampel adalah 1 permintaan per detik dan 5 persen dari permintaan tambahan. Anda tidak dapat mengonfigurasi laju X-Ray pengambilan sampel untuk fungsi Anda.

Menggunakan Powertools untuk AWS Lambda (Python) dan AWS CDK untuk menelusuri

Ikuti langkah-langkah di bawah ini untuk mengunduh, membangun, dan menyebarkan contoh aplikasi Hello World Python dengan Powertools terintegrasi untuk modul AWS Lambda (Python) menggunakan modul. AWS CDK Aplikasi ini mengimplementasikan backend API dasar dan menggunakan Powertools untuk memancarkan log, metrik, dan jejak. Ini terdiri dari titik akhir Amazon API Gateway dan Fungsi Peristiwa Lambda. Saat Anda mengirim permintaan GET ke titik akhir API Gateway, fungsi tersebut memanggil, mengirim log dan metrik menggunakan Format Metrik Tertanam ke CloudWatch, dan mengirimkan jejak ke. AWS X-Ray Fungsi mengembalikan pesan hello world.

Prasyarat

Untuk menyelesaikan langkah-langkah di bagian ini, Anda harus memiliki hal-hal berikut:

- Python 3.11
- AWS CLI versi 2
- AWS CDK versi 2
- AWS SAM CLI versi 1.75 atau yang lebih baru. Jika Anda memiliki versi CLI yang lebih lama, lihat Memutakhirkan AWS SAM CLI. AWS SAM

Menyebarkan sampel AWS CDK application

1. Buat direktori proyek untuk aplikasi baru Anda.

```
mkdir hello-world
cd hello-world
```

2. Inisialisasi aplikasi.

```
cdk init app --language python
```

3. Instal dependensi Python.

```
pip install -r requirements.txt
```

4. Buat direktori lambda_function di bawah folder root.

```
mkdir lambda_function
```

```
cd lambda_function
```

5. Buat file app.py dan tambahkan kode berikut ke file. Ini adalah kode untuk fungsi tersebut.

```
from aws_lambda_powertools.event_handler import APIGatewayRestResolver
from aws_lambda_powertools.utilities.typing import LambdaContext
from aws_lambda_powertools.logging import correlation_paths
from aws_lambda_powertools import Logger
from aws_lambda_powertools import Tracer
from aws_lambda_powertools import Metrics
from aws_lambda_powertools.metrics import MetricUnit

app = APIGatewayRestResolver()
tracer = Tracer()
logger = Logger()
metrics = Metrics(namespace="PowertoolsSample")

@app.get("/hello")
@tracer.capture_method
def hello():
    # adding custom metrics
    # See: https://docs.powertools.aws.dev/lambda-python/latest/core/metrics/
    metrics.add_metric(name="HelloWorldInvocations", unit=MetricUnit.Count,
 value=1)

    # structured log
    # See: https://docs.powertools.aws.dev/lambda-python/latest/core/logger/
    logger.info("Hello world API - HTTP 200")
    return {"message": "hello world"}

# Enrich logging with contextual information from Lambda
@logger.inject_lambda_context(correlation_id_path=correlation_paths.API_GATEWAY_REST)
# Adding tracer
# See: https://docs.powertools.aws.dev/lambda-python/latest/core/tracer/
@tracer.capture_lambda_handler
# ensures metrics are flushed upon request completion/failure and capturing
 ColdStart metric
@metrics.log_metrics(capture_cold_start_metric=True)
def lambda_handler(event: dict, context: LambdaContext) -> dict:
    return app.resolve(event, context)
```

6. Buka direktori hello_world. Anda akan melihat file bernama hello_world_stack.py.

```
cd ..
cd hello_world
```

7. Buka hello_world_stack.py dan tambahkan kode berikut ke file. Ini berisi Konstruktor Lambda, yang membuat fungsi, mengonfigurasi variabel lingkungan untuk Powertools dan menetapkan retensi log ke satu minggu, dan Konstruktor ApiGatewayv 1, yang membuat REST API.

```
from aws_cdk import (
    Stack,
    aws_apigateway as apigwv1,
    aws_lambda as lambda_,
    CfnOutput,
    Duration
)
from constructs import Construct

class HelloWorldStack(Stack):

    def __init__(self, scope: Construct, construct_id: str, **kwargs) -> None:
        super().__init__(scope, construct_id, **kwargs)

        # Powertools Lambda Layer
        powertools_layer = lambda_.LayerVersion.from_layer_version_arn(
            self,
            id="lambda-powertools",
            # At the moment we wrote this example, the aws_lambda_python_alpha CDK
 constructor is in Alpha, o we use layer to make the example simpler
            # See https://docs.aws.amazon.com/cdk/api/v2/python/
aws_cdk.aws_lambda_python_alpha/README.html
            # Check all Powertools layers versions here: https://
docs.powertools.aws.dev/lambda-python/latest/#lambda-layer
            layer_version_arn=f"arn:aws:lambda:
{self.region}:017000801446:layer:AWSLambdaPowertoolsPythonV2:21"
        )

        function = lambda_.Function(self,
            'sample-app-lambda',
            runtime=lambda_.Runtime.PYTHON_3_11,
            layers=[powertools_layer],
            code = lambda_.Code.from_asset("./lambda_function/"),
            handler="app.lambda_handler",
            memory_size=128,
```

```
            timeout=Duration.seconds(3),
            architecture=lambda_.Architecture.X86_64,
            environment={
                "POWERTOOLS_SERVICE_NAME": "PowertoolsHelloWorld",
                "POWERTOOLS_METRICS_NAMESPACE": "PowertoolsSample",
                "LOG_LEVEL": "INFO"
            }
        )

        apigw = apigwv1.RestApi(self, "PowertoolsAPI",
 deploy_options=apigwv1.StageOptions(stage_name="dev"))

        hello_api = apigw.root.add_resource("hello")
        hello_api.add_method("GET", apigwv1.LambdaIntegration(function,
 proxy=True))

        CfnOutput(self, "apiUrl", value=f"{apigw.url}hello")
```

8. Men-deploy aplikasi Anda.

```
cd ..
cdk deploy
```

9. Dapatkan URL aplikasi yang digunakan:

```
aws cloudformation describe-stacks --stack-name HelloWorldStack --query
 'Stacks[0].Outputs[?OutputKey==`apiUrl`].OutputValue' --output text
```

10. Memanggil titik akhir API:

```
curl -X GET <URL_FROM_PREVIOUS_STEP>
```

Jika berhasil, Anda akan melihat tanggapan ini:

```
{"message":"hello world"}
```

11. Untuk mendapatkan jejak untuk fungsi tersebut, jalankan jejak sam.

```
sam traces
```

Output jejak terlihat seperti ini:

```
New XRay Service Graph
  Start time: 2023-02-03 14:59:50+00:00
  End time: 2023-02-03 14:59:50+00:00
  Reference Id: 0 - (Root) AWS::Lambda - sam-app-HelloWorldFunction-YBg8yfYtOc9j -
 Edges: [1]
   Summary_statistics:
     - total requests: 1
     - ok count(2XX): 1
     - error count(4XX): 0
     - fault count(5XX): 0
     - total response time: 0.924
  Reference Id: 1 - AWS::Lambda::Function - sam-app-HelloWorldFunction-YBg8yfYtOc9j
 - Edges: []
   Summary_statistics:
     - total requests: 1
     - ok count(2XX): 1
     - error count(4XX): 0
     - fault count(5XX): 0
     - total response time: 0.016
  Reference Id: 2 - client - sam-app-HelloWorldFunction-YBg8yfYtOc9j - Edges: [0]
   Summary_statistics:
     - total requests: 0
     - ok count(2XX): 0
     - error count(4XX): 0
     - fault count(5XX): 0
     - total response time: 0

XRay Event [revision 1] at (2023-02-03T14:59:50.204000) with id
 (1-63dd2166-434a12c22e1307ff2114f299) and duration (0.924s)
 - 0.924s - sam-app-HelloWorldFunction-YBg8yfYtOc9j [HTTP: 200]
 - 0.016s - sam-app-HelloWorldFunction-YBg8yfYtOc9j
   - 0.739s - Initialization
   - 0.016s - Invocation
     - 0.013s - ## lambda_handler
       - 0.000s - ## app.hello
   - 0.000s - Overhead
```

12. Ini adalah titik akhir API publik yang dapat diakses melalui internet. Kami menyarankan Anda menghapus titik akhir setelah pengujian.

```
cdk destroy
```

Menggunakan ADOT untuk instrumen fungsi Python Anda

ADOT menyediakan lapisan Lambda yang dikelola sepenuhnya yang mengemas semua yang Anda butuhkan untuk mengumpulkan data telemetri menggunakan OTel SDK. Dengan menggunakan lapisan ini, Anda dapat menginstruksikan fungsi Lambda Anda tanpa harus memodifikasi kode fungsi apa pun. Anda juga dapat mengonfigurasi lapisan Anda untuk melakukan inisialisasi khusus OTel. Untuk informasi selengkapnya, lihat Konfigurasi khusus untuk Kolektor ADOT di Lambda dalam dokumentasi ADOT.

Untuk runtime Python, Anda dapat menambahkan layer AWS Lambda terkelola untuk ADOT Python untuk secara otomatis instrumen fungsi Anda. Lapisan ini berfungsi untuk arsitektur arm64 dan x86_64. Untuk petunjuk rinci tentang cara menambahkan layer ini, lihat AWS Distro untuk OpenTelemetry Lambda Support for Python dalam dokumentasi ADOT.

Menggunakan X-Ray SDK untuk instrumen fungsi Python Anda

Untuk merekam detail tentang panggilan yang dilakukan fungsi Anda ke sumber daya lain dalam aplikasi Anda, Anda juga dapat menggunakan AWS X-Ray SDK for Python. Untuk mendapatkan SDK, tambahkan paket `aws-xray-sdk` ke dependensi aplikasi Anda.

Example requirements.txt

```
jsonpickle==1.3
aws-xray-sdk==2.4.3
```

Dalam kode fungsi Anda, Anda dapat menginstruksikan klien AWS SDK dengan menambal `boto3` perpustakaan dengan modul. `aws_xray_sdk.core`

Example fungsi — Menelusuri AWS Klien SDK

```
import boto3
from aws_xray_sdk.core import xray_recorder
from aws_xray_sdk.core import patch_all

logger = logging.getLogger()
logger.setLevel(logging.INFO)
patch_all()

client = boto3.client('lambda')
client.get_account_settings()
```

```
def lambda_handler(event, context):
    logger.info('## ENVIRONMENT VARIABLES\r' + jsonpickle.encode(dict(**os.environ)))
  ...
```

Setelah Anda menambahkan dependensi yang benar dan membuat perubahan kode yang diperlukan, aktifkan penelusuran dalam konfigurasi fungsi Anda melalui konsol Lambda atau API.

Mengaktifkan penelusuran dengan konsol Lambda

Untuk mengaktifkan penelusuran aktif di Fungsi Peristiwa Lambda Anda dengan konsol, ikuti langkah-langkah berikut:

Untuk mengaktifkan penelusuran aktif

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih Fungsi Acara.
3. Pilih Konfigurasi dan kemudian pilih Alat Pemantauan dan operasi.
4. Di bawah Alat pemantauan tambahan, pilih Edit.
5. Di bawah Sinyal CloudWatch Aplikasi dan AWS X-Ray, pilih Aktifkan untuk jejak layanan Lambda.
6. Pilih Simpan.

Mengaktifkan penelusuran dengan Lambda API

Konfigurasikan penelusuran pada fungsi Lambda Anda dengan AWS CLI AWS atau SDK, gunakan operasi API berikut:

- UpdateFunctionConfiguration
- GetFunctionConfiguration
- CreateFunction

Contoh AWS CLI perintah berikut memungkinkan penelusuran aktif pada fungsi bernama my-function.

```
aws lambda update-function-configuration --function-name my-function \
--tracing-config Mode=Active
```

Mode penelusuran adalah bagian dari konfigurasi khusus versi saat Anda mempublikasikan versi fungsi Anda. Anda tidak dapat mengubah mode pelacakan pada versi yang telah diterbitkan.

Mengaktifkan penelusuran dengan CloudFormation

Untuk mengaktifkan penelusuran pada `AWS::Lambda::Function` sumber daya dalam CloudFormation templat, gunakan `TracingConfig` properti.

Example fungsi-inline.yml. — Konfigurasi penelusuran

```
Resources:
  function:
    Type: AWS::Lambda::Function
    Properties:
      TracingConfig:
        Mode: Active
      ...
```

Untuk sumber `AWS::Serverless::Function` daya AWS Serverless Application Model (AWS SAM), gunakan `Tracing` properti.

Example template.yml — Konfigurasi penelusuran

```
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      Tracing: Active
      ...
```

Menafsirkan jejak X-Ray

Fungsi Acara Anda memerlukan izin untuk mengunggah data jejak ke X-Ray. Saat Anda mengaktifkan penelusuran di konsol Lambda, Lambda menambahkan izin yang diperlukan ke peran eksekusi Fungsi Acara Anda. Atau, tambahkan kebijakan AWSXRayDaemonWriteAccess ke peran eksekusi.

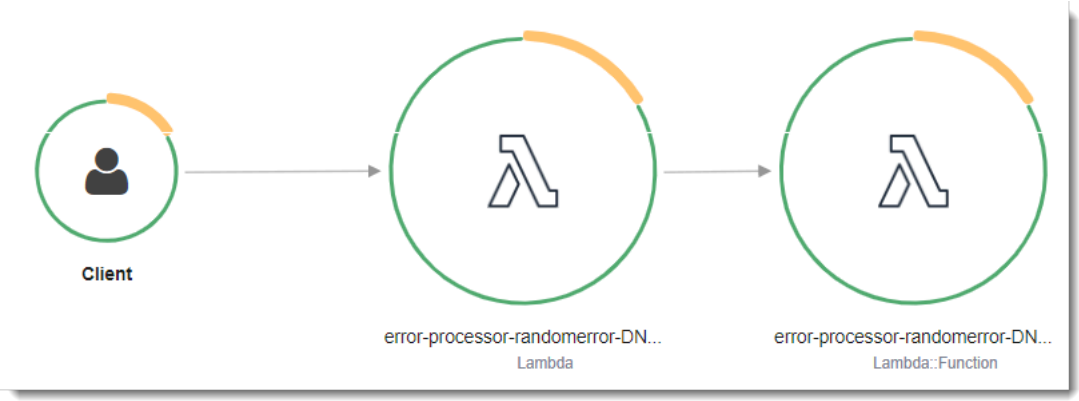
Setelah mengonfigurasi penelusuran aktif, Anda dapat mengamati permintaan tertentu melalui aplikasi Anda. Grafik X-Ray layanan menunjukkan informasi tentang aplikasi Anda dan semua komponennya. Contoh berikut menunjukkan aplikasi dengan dua fungsi. Fungsi utama memproses

kejadian dan terkadang mengembalikan kesalahan. Fungsi kedua di bagian atas memproses kesalahan yang muncul di grup log pertama dan menggunakan AWS SDK untuk memanggil X-Ray, Amazon Simple Storage Service (Amazon S3) Service, dan Amazon Logs. CloudWatch



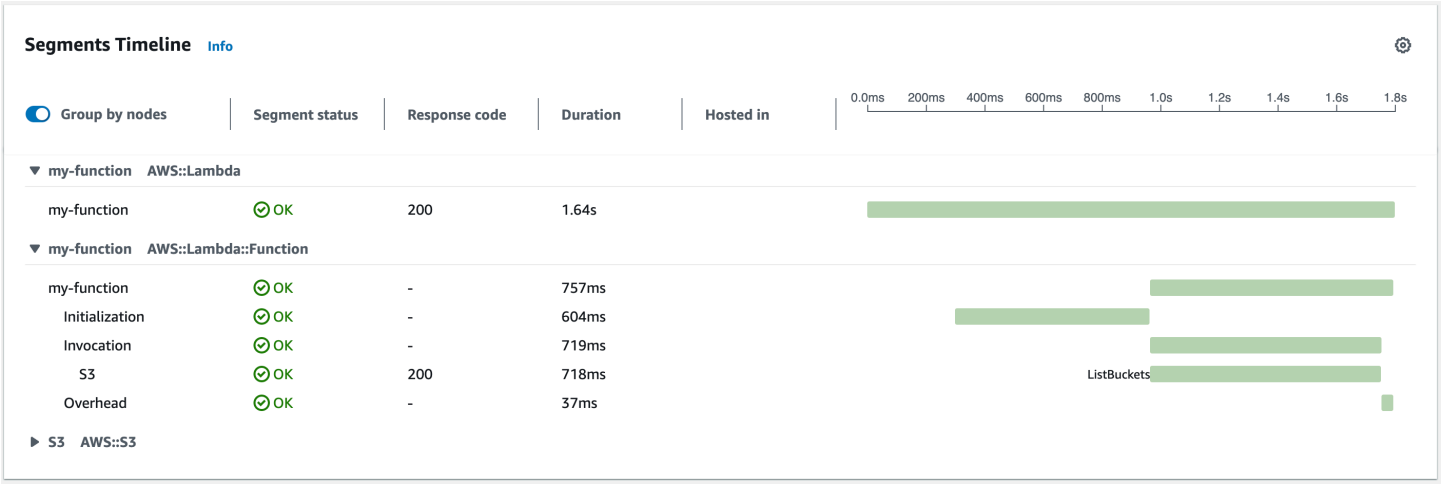
X-Ray tidak melacak semua permintaan ke aplikasi Anda. X-Ray menerapkan algoritma pengambilan sampel untuk memastikan bahwa penelusuran efisien, sambil tetap memberikan sampel yang representatif dari semua permintaan. Tingkat pengambilan sampel adalah 1 permintaan per detik dan 5 persen dari permintaan tambahan. Anda tidak dapat mengonfigurasi laju X-Ray pengambilan sampel untuk fungsi Anda.

Di X-Ray, jejak mencatat informasi tentang permintaan yang diproses oleh satu atau beberapa layanan. Lambda mencatat 2 segmen per jejak, yang menciptakan dua node pada grafik layanan. Gambar berikut menyoroti dua node ini:



Node pertama di sebelah kiri mewakili layanan Lambda, yang menerima permintaan pemanggilan. Node kedua mewakili fungsi Lambda spesifik Anda. Contoh berikut menunjukkan jejak dengan dua segmen ini. Keduanya bernama fungsi saya, tetapi yang satu memiliki asal `AWS::Lambda`

dan yang lainnya memiliki asal usul. `AWS::Lambda::Function` Jika `AWS::Lambda` segmen menunjukkan kesalahan, layanan Lambda mengalami masalah. Jika `AWS::Lambda::Function` segmen menunjukkan kesalahan, fungsi Anda mengalami masalah.



Contoh ini memperluas `AWS::Lambda::Function` segmen untuk menunjukkan tiga subsegmennya.

> **Note**
>
> AWS saat ini menerapkan perubahan pada layanan Lambda. Karena perubahan ini, Anda mungkin melihat perbedaan kecil antara struktur dan konten pesan log sistem dan segmen jejak yang dipancarkan oleh Fungsi Peristiwa Lambda yang berbeda di Anda. Akun AWS Contoh jejak yang ditunjukkan di sini menggambarkan segmen fungsi gaya lama. Perbedaan antara segmen gaya lama dan baru dijelaskan dalam paragraf berikut.
> Perubahan ini akan diterapkan selama beberapa minggu mendatang, dan semua fungsi di semua Wilayah AWS kecuali China GovCloud dan wilayah akan bertransisi untuk menggunakan pesan log format baru dan segmen pelacakan.

Segmen fungsi gaya lama berisi subsegmen berikut:

- Inisialisasi – Mewakili waktu yang dihabiskan untuk memuat fungsi dan menjalankan kode inisialisasi. Subsegmen ini hanya muncul untuk peristiwa pertama yang diproses oleh setiap instance fungsi Anda.
- Doa - Merupakan waktu yang dihabiskan untuk menjalankan kode handler Anda.
- Overhead - Merupakan waktu yang dihabiskan runtime Lambda untuk mempersiapkan diri untuk menangani acara berikutnya.

Segmen fungsi gaya baru tidak berisi subsegmen. `Invocation` Sebagai gantinya, subsegmen pelanggan dilampirkan langsung ke segmen fungsi. Untuk informasi lebih lanjut tentang struktur segmen fungsi gaya lama dan baru, lihat. the section called “Memahami X-Ray jejak”

Anda juga dapat melakukan instrumentasi klien HTTP, merekam kueri SQL, dan membuat subsegmen khusus dengan anotasi dan metadata. Untuk informasi selengkapnya, lihat AWS X-Ray SDK for Pythondi Panduan AWS X-Ray Pengembang.

> **Harga**
>
> Anda dapat menggunakan X-Ray tracing gratis setiap bulan hingga batas tertentu sebagai bagian dari Tingkat AWS Gratis. Di luar ambang batas itu, X-Ray biaya untuk penyimpanan dan pengambilan jejak. Untuk informasi selengkapnya, lihat harga AWS X-Ray.

Menyimpan dependensi runtime dalam lapisan (SDK) X-Ray

Jika Anda menggunakan X-Ray SDK untuk menginstruksikan klien AWS SDK kode fungsi Anda, paket penerapan Anda bisa menjadi cukup besar. Untuk menghindari mengunggah dependensi runtime setiap kali Anda memperbarui kode fungsi, paketkan X-Ray SDK dalam lapisan Lambda.

Contoh berikut menunjukkan `AWS::Serverless::LayerVersion` sumber daya yang menyimpan file AWS X-Ray SDK for Python.

Example template.yml - Lapisan Dependensi

```
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: function/.
      Tracing: Active
      Layers:
        - !Ref libs
      ...
  libs:
    Type: AWS::Serverless::LayerVersion
    Properties:
      LayerName: blank-python-lib
      Description: Dependencies for the blank-python sample app.
      ContentUri: package/.
```

```
      CompatibleRuntimes:
        - python3.11
```

Dengan konfigurasi ini, Anda memperbarui lapisan pustaka hanya jika Anda mengubah dependensi runtime Anda. Karena paket penerapan fungsi hanya berisi kode Anda, ini dapat membantu mengurangi waktu upload.

Membuat lapisan untuk dependensi memerlukan perubahan build untuk menghasilkan arsip lapisan sebelum penerapan. Untuk contoh pekerjaan, lihat aplikasi sampel blank-python.

Membangun Fungsi Acara Lambda dengan Ruby

Anda dapat menjalankan kode Ruby di Fungsi AWS Lambda Acara. Lambda menyediakan runtime untuk Ruby yang menjalankan kode Anda untuk memproses peristiwa. Kode Anda berjalan di lingkungan yang menyertakan AWS SDK untuk Ruby, dengan kredensi dari peran AWS Identity and Access Management (IAM) yang Anda kelola. Untuk mempelajari lebih lanjut tentang versi SDK yang disertakan dengan runtime Ruby, lihat. the section called “Runtime-included Versi SDK”

Lambda mendukung runtime Ruby berikut.

Nama	Pengidentifikasi	Sistem operasi	Tanggal pengusangan	Buat fungsi blok	Pembaruan fungsi blok
Ruby 4.0	`ruby4.0`	Amazon Linux 2023	31 Mar 2029	30 Apr 2029	31 Mei 2029
Ruby 3.4	`ruby3.4`	Amazon Linux 2023	31 Mar 2028	30 Apr 2028	31 Mei 2028
Ruby 3.3	`ruby3.3`	Amazon Linux 2023	31 Mar 2027	Jul 29, 2027	Agustus 31, 2027

Untuk membuat Ruby Event Function

1. Buka Konsol Lambda.
2. Pilih Buat Fungsi Acara.
3. Konfigurasikan pengaturan berikut:
   - Nama fungsi: Masukkan nama untuk fungsi tersebut.

- Runtime: Pilih Ruby 4.0.

4. Pilih Buat Fungsi Acara.

Konsol membuat Fungsi Peristiwa Lambda dengan satu file sumber bernama. `lambda_function.rb` Anda dapat mengedit file ini dan menambahkan lebih banyak file di editor kode bawaan. Di bagian DEPLOY, pilih Deploy untuk memperbarui kode fungsi Anda. Kemudian, untuk menjalankan kode Anda, pilih Create test event di bagian TEST EVENTS.

File `lambda_function.rb` mengekspor fungsi bernama `lambda_handler` yang menerima objek peristiwa dan objek konteks. Ini adalah fungsi handler yang dipanggil Lambda saat fungsi tersebut dipanggil. Runtime Ruby Event Function mendapatkan event pemanggilan dari Lambda dan meneruskannya ke handler. Dalam konfigurasi fungsi, nilai handler adalah `lambda_function.lambda_handler`.

Saat Anda menyimpan kode fungsi, konsol Lambda membuat paket penyebaran arsip file.zip. Saat Anda mengembangkan kode fungsi di luar konsol (menggunakan IDE), Anda perlu membuat paket penerapan untuk mengunggah kode Anda ke fungsi Lambda.

Runtime fungsi meneruskan objek konteks ke handler, selain peristiwa invokasi. Objek konteks berisi informasi tambahan tentang lingkungan invokasi, fungsi, dan eksekusi. Informasi selengkapnya tersedia dari variabel lingkungan.

Fungsi Lambda Anda dilengkapi dengan grup CloudWatch log Log. Runtime fungsi mengirimkan detail tentang setiap invokasi ke CloudWatch Logs. Detail tersebut menyampaikan log yang dihasilkan fungsi Anda selama invokasi. Jika fungsi mengembalikan kesalahan, Lambda memformat kesalahan dan mengembalikannya ke pemanggil.

Topik

- Runtime-included Versi SDK
- Mengaktifkan Ruby JIT Lainnya (YJIT)
- Tentukan penangan Fungsi Peristiwa Lambda di Ruby
- Menyebarkan Fungsi Acara Ruby Lambda dengan arsip file.zip
- Menyebarkan Fungsi Acara Ruby Lambda dengan gambar kontainer
- Bekerja dengan lapisan untuk Fungsi Acara Ruby Lambda
- Menggunakan objek konteks Lambda untuk mengambil informasi Fungsi Acara Ruby
- Log dan pantau Fungsi Acara Ruby Lambda

- Menginstrumentasi kode Ruby di AWS Lambda

Runtime-included Versi SDK

Versi AWS SDK yang disertakan dalam runtime Ruby bergantung pada versi runtime dan versi Anda. Wilayah AWS AWS SDK for Ruby dirancang untuk modular dan dipisahkan oleh. Layanan AWS Untuk menemukan nomor versi permata layanan tertentu yang disertakan dalam runtime yang Anda gunakan, buat Fungsi Peristiwa Lambda dengan kode dalam format berikut. Ganti `aws-sdk-s3` dan `Aws::S3` dengan nama permata layanan yang digunakan kode Anda.

```
require 'aws-sdk-s3'

def lambda_handler(event:, context:)
  puts "Service gem version: #{Aws::S3::GEM_VERSION}"
  puts "Core version: #{Aws::CORE_GEM_VERSION}"
end
```

Mengaktifkan Ruby JIT Lainnya (YJIT)

Runtime Ruby mendukung YJIT, kompiler Ruby JIT yang ringan dan minimalis. YJIT memberikan kinerja yang jauh lebih tinggi, tetapi juga menggunakan lebih banyak memori daripada penerjemah Ruby. YJIT direkomendasikan untuk beban kerja Ruby on Rails.

YJIT tidak diaktifkan secara default. Untuk mengaktifkan YJIT untuk fungsi Ruby, atur variabel `RUBY_YJIT_ENABLE` lingkungan ke. `1` Untuk mengonfirmasi bahwa YJIT diaktifkan, cetak hasil metode. `RubyVM::YJIT.enabled?`

Example— Konfirmasikan bahwa YJIT diaktifkan

```
puts(RubyVM::YJIT.enabled?())
# => true
```

Tentukan penangan Fungsi Peristiwa Lambda di Ruby

Lambda Event Function handler adalah metode dalam kode fungsi Anda yang memproses peristiwa. Saat fungsi Anda diaktifkan, Lambda menjalankan metode handler. Fungsi Anda berjalan sampai handler mengembalikan respons, keluar, atau waktu habis.

Topik

- Dasar-dasar penangan Ruby
- Praktik terbaik kode untuk Fungsi Acara Ruby Lambda

Dasar-dasar penangan Ruby

Dalam contoh berikut, file `function.rb` menentukan metode handler bernama `handler`. Fungsi handler mengambil dua objek sebagai input dan mengembalikan dokumen JSON.

Example function.rb

```
require 'json'

def handler(event:, context:)
    { event: JSON.generate(event), context: JSON.generate(context.inspect) }
end
```

Dalam konfigurasi fungsi Anda, pengaturan `handler` memberi tahu Lambda tempat untuk menemukan penangan. Untuk contoh sebelumnya, nilai yang benar untuk pengaturan ini adalah **`function.handler`**. Ini mencakup dua nama yang dipisahkan oleh titik: nama file dan metode handler.

Anda juga dapat menentukan metode penangan dalam kelas. Contoh berikut mendefinisikan metode penangan bernama `process` pada kelas bernama `Handler` dalam modul bernama `LambdaFunctions`.

Example source.rb

```
module LambdaFunctions
  class Handler
    def self.process(event:,context:)
      "Hello!"
    end
```

```
  end
end
```

Dalam hal ini, pengaturan penangan adalah **`source.LambdaFunctions::Handler.process`**.

Dua objek yang diterima handler adalah kejadian dan konteks invokasi. Kejadian tersebut adalah objek Ruby yang berisi muatan yang disediakan oleh pelaku invokasi. Jika muatannya adalah dokumen JSON, objek kejadiannya adalah hash Ruby. Jika tidak, objek kejadiannya adalah string. Objek konteks memiliki metode dan properti yang menyediakan informasi tentang invokasi, fungsi, dan lingkungan eksekusi.

Penangan fungsi dijalankan setiap kali fungsi Lambda Anda diinvokasi. Kode statis di luar penangan dijalankan satu kali per instans fungsi. Jika penangan menggunakan sumber daya, seperti klien SDK dan koneksi database, Anda dapat membuatnya di luar metode penangan untuk menggunakannya kembali untuk beberapa invokasi.

Setiap instans fungsi Anda dapat memproses beberapa kejadian invokasi, tetapi hanya memproses satu kejadian dalam satu waktu. Jumlah instans yang memproses kejadian pada waktu tertentu adalah konkurensi fungsi Anda. Untuk informasi lebih lanjut tentang lingkungan eksekusi Lambda, lihat Memahami siklus hidup lingkungan pelaksanaan Lambda.

Praktik terbaik kode untuk Fungsi Acara Ruby Lambda

Patuhi pedoman dalam daftar berikut untuk menggunakan praktik pengkodean terbaik saat membangun fungsi Lambda Anda:

- Pisahkan handler Lambda dari logika inti Anda. Ini membuatnya lebih mudah untuk membuat fungsi yang lebih dapat diuji unit. Misalnya, di Ruby, ini mungkin terlihat seperti:

```
def lambda_handler(event:, context:)
    foo = event['foo']
    bar = event['bar']

    result = my_lambda_function(foo:, bar:)

end

def my_lambda_function(foo:, bar:)
    // MyLambdaFunction logic here

end
```

- Kontrol dependensi dalam paket penerapan fungsi Anda. Lingkungan AWS Lambda eksekusi berisi sejumlah pustaka. Untuk runtime Ruby, ini termasuk SDK AWS .

  Untuk mengaktifkan serangkaian fitur dan pembaruan keamanan terbaru, Lambda memperbarui pustaka ini secara berkala. Pembaruan ini mungkin memperkenalkan perubahan halus pada perilaku fungsi Lambda Anda. Untuk memiliki kendali penuh atas dependensi yang digunakan fungsi Anda, kemas semua dependensi Anda dengan paket deployment Anda.
- Minimalkan kompleksitas dependensi Anda. Utamakan memilih kerangka kerja lebih sederhana yang cepat dimuat dalam memulai lingkungan eksekusi.
- Minimalkan ukuran paket penerapan Anda sesuai kebutuhan runtime-nya. Ini mengurangi jumlah waktu yang diperlukan untuk paket penerapan Anda untuk diunduh dan dibongkar sebelum pemanggilan. Untuk fungsi yang ditulis di Ruby, hindari mengunggah seluruh pustaka AWS SDK sebagai bagian dari paket penerapan Anda. Sebagai gantinya, secara selektif bergantung pada permata yang mengambil komponen SDK yang Anda butuhkan (misalnya, permata DynamoDB atau Amazon S3 SDK).

Manfaatkan penggunaan kembali lingkungan eksekusi untuk meningkatkan kinerja fungsi Anda. Inisialisasi klien SDK dan koneksi basis data di luar fungsi handler, dan lakukan caching aset statis secara lokal di direktori `/tmp`. Invokasi selanjutnya yang diproses oleh instans yang sama dari fungsi Anda dapat menggunakan kembali sumber daya ini. Ini menghemat biaya dengan mengurangi waktu pengoperasian fungsi.

Untuk menghindari potensi kebocoran data di seluruh invokasi, jangan menggunakan lingkungan eksekusi untuk menyimpan data pengguna, peristiwa, atau informasi lainnya implikasi keamanan. Jika fungsi Anda bergantung pada status yang dapat disenyapkan yang tidak dapat disimpan dalam memori di dalam handler, pertimbangkan untuk membuat fungsi terpisah atau versi terpisah dari fungsi untuk setiap pengguna.

Gunakan arahan keep-alive untuk mempertahankan koneksi yang persisten. Lambda membersihkan koneksi idle dari waktu ke waktu. Mencoba menggunakan ulang koneksi idle saat mengidentifikasi suatu fungsi akan menyebabkan kesalahan koneksi. Untuk mempertahankan koneksi yang persisten, gunakan arahan tetap aktif yang berkaitan dengan runtime Anda. Sebagai contoh, lihat Menggunakan Kembali Koneksi dengan Keep-Alive in Node.js.

Gunakan variabel lingkungan untuk meneruskan parameter operasional ke fungsi Anda. Misalnya, jika Anda ingin menulis ke bucket Amazon S3 alih-alih melakukan hard-coding nama bucket yang Anda tulis, konfigurasikan nama bucket sebagai variabel lingkungan.

Hindari menggunakan pemanggilan rekursif dalam fungsi Lambda Anda, di mana fungsi memanggil dirinya sendiri atau memulai proses yang dapat memanggil fungsi lagi. Hal ini dapat menyebabkan volume invokasi fungsi yang tidak diinginkan dan peningkatan biaya. Jika Anda melihat volume pemanggilan yang tidak diinginkan, setel konkurensi fungsi cadangan untuk `0` segera membatasi semua pemanggilan ke fungsi, saat Anda memperbarui kode.

Jangan gunakan API non-publik yang tidak terdokumentasi dalam kode fungsi Lambda Anda. Untuk runtime AWS Lambda terkelola, Lambda secara berkala menerapkan pembaruan keamanan dan fungsional ke API internal Lambda. Pembaruan API internal ini mungkin tidak kompatibel ke belakang, yang menyebabkan konsekuensi yang tidak diinginkan seperti kegagalan pemanggilan jika fungsi Anda memiliki ketergantungan pada API non-publik ini. Lihat referensi API untuk daftar API yang tersedia untuk umum.

Tulis kode idempoten. Menulis kode idempoten untuk fungsi Anda memastikan bahwa peristiwa duplikat ditangani dengan cara yang sama. Kode Anda harus memvalidasi peristiwa dengan benar dan menangani peristiwa duplikat dengan anggun. Untuk informasi selengkapnya, lihat Bagaimana cara membuat fungsi Lambda saya idempoten? .

Menyebarkan Fungsi Acara Ruby Lambda dengan arsip file.zip

Kode AWS Lambda fungsi Anda terdiri dari file.rb yang berisi kode handler fungsi Anda, bersama dengan dependensi tambahan (permata) yang bergantung pada kode Anda. Untuk menyebarkan kode fungsi ini ke Lambda, Anda menggunakan paket penerapan. Paket ini dapat berupa arsip file.zip atau gambar kontainer. Untuk informasi selengkapnya tentang penggunaan gambar kontainer dengan Ruby, lihat Menerapkan Fungsi Acara Ruby Lambda dengan gambar kontainer.

Untuk membuat paket penyebaran Anda sebagai arsip file.zip, Anda dapat menggunakan utilitas arsip file.zip bawaan alat baris perintah Anda, atau utilitas file.zip lainnya seperti 7zip. Contoh yang ditampilkan di bagian berikut mengasumsikan Anda menggunakan `zip` alat baris perintah di lingkungan Linux atau macOS. Untuk menggunakan perintah yang sama di Windows, Anda dapat menginstal Subsistem Windows untuk Linux untuk mendapatkan Windows-integrated versi Ubuntu dan Bash.

Perhatikan bahwa Lambda menggunakan izin file POSIX, jadi Anda mungkin perlu mengatur izin untuk folder paket penyebaran sebelum membuat arsip file.zip.

Perintah contoh di bagian berikut menggunakan utilitas Bundler untuk menambahkan dependensi ke paket penyebaran Anda. Untuk menginstal bundler, jalankan perintah berikut.

```
gem install bundler
```

Bagian-bagian

- Dependensi di Ruby
- Membuat paket penerapan.zip tanpa dependensi
- Membuat paket penerapan.zip dengan dependensi
- Membuat layer Ruby untuk dependensi Anda
- Membuat paket penerapan.zip dengan pustaka asli
- Membuat dan memperbarui Fungsi Acara Ruby Lambda menggunakan file.zip

Dependensi di Ruby

Untuk Lambda Event Functions yang menggunakan runtime Ruby, dependensi dapat berupa permata Ruby. Ketika Anda menerapkan fungsi Anda menggunakan arsip.zip, Anda dapat menambahkan dependensi ini ke file.zip Anda dengan kode fungsi Anda atau menggunakan lapisan

Lambda. Lapisan adalah file.zip terpisah yang dapat berisi kode tambahan dan konten lainnya. Untuk mempelajari lebih lanjut tentang menggunakan layer Lambda, lihat. the section called “Lapisan”

Runtime Ruby termasuk. AWS SDK untuk Ruby Jika fungsi Anda menggunakan SDK, Anda tidak perlu menggabungkannya dengan kode Anda. Namun, untuk mempertahankan kontrol penuh atas dependensi Anda, atau menggunakan versi SDK tertentu, Anda dapat menambahkannya ke paket penerapan fungsi Anda.

Anda dapat menyertakan SDK dalam file.zip Anda, atau menambahkannya menggunakan lapisan Lambda. Dependensi di file.zip Anda atau di lapisan Lambda lebih diutamakan daripada versi yang disertakan dalam runtime. Untuk mengetahui versi SDK for Ruby mana yang disertakan dalam versi runtime Anda, lihat. the section called “Runtime-included Versi SDK”

Di bawah model tanggung jawab AWS bersama, Anda bertanggung jawab atas pengelolaan dependensi apa pun dalam paket penerapan fungsi Anda. Ini termasuk menerapkan pembaruan dan patch keamanan. Untuk memperbarui dependensi dalam paket penerapan fungsi Anda, pertama buat file.zip baru dan kemudian unggah ke Lambda. Lihat Membuat paket penerapan.zip dengan dependensi dan Membuat dan memperbarui Fungsi Acara Ruby Lambda menggunakan file.zip untuk informasi lebih lanjut.

Membuat paket penerapan.zip tanpa dependensi

Jika kode fungsi Anda tidak memiliki dependensi, file.zip Anda hanya berisi file.rb dengan kode handler fungsi Anda. Gunakan utilitas zip pilihan Anda untuk membuat file.zip dengan file.rb Anda di root. Jika file.rb tidak berada di root file.zip Anda, Lambda tidak akan dapat menjalankan kode Anda.

Untuk mempelajari cara menerapkan file.zip Anda untuk membuat fungsi Lambda baru atau memperbarui yang sudah ada, lihat. Membuat dan memperbarui Fungsi Acara Ruby Lambda menggunakan file.zip

Membuat paket penerapan.zip dengan dependensi

Jika kode fungsi Anda bergantung pada permata Ruby tambahan, Anda dapat menambahkan dependensi ini ke file.zip Anda dengan kode fungsi Anda atau menggunakan lapisan Lambda. Petunjuk di bagian ini menunjukkan kepada Anda cara memasukkan dependensi dalam paket penerapan.zip Anda. Untuk petunjuk tentang cara memasukkan dependensi Anda dalam lapisan, lihat. the section called “Membuat layer Ruby untuk dependensi Anda”

Misalkan kode fungsi Anda disimpan dalam file bernama `lambda_function.rb` di direktori proyek Anda. Contoh perintah CLI berikut membuat file.zip bernama `my_deployment_package.zip` berisi kode fungsi Anda dan dependensinya.

Untuk membuat paket deployment

1. Di direktori proyek Anda, buat a `Gemfile` untuk menentukan dependensi Anda di.

   ```
   bundle init
   ```

2. Menggunakan editor teks pilihan Anda, edit `Gemfile` untuk menentukan dependensi fungsi Anda. Misalnya, untuk menggunakan permata TZInfo, edit Anda `Gemfile` agar terlihat seperti berikut ini.

   ```
   source "https://rubygems.org"
   gem "tzinfo"
   ```

3. Jalankan perintah berikut untuk menginstal permata yang ditentukan `Gemfile` dalam direktori proyek Anda. Perintah ini ditetapkan `vendor/bundle` sebagai jalur default untuk instalasi permata.

   ```
   bundle config set --local path 'vendor/bundle' && bundle install
   ```

   Anda akan melihat output seperti yang berikut ini.

   ```
   Fetching gem metadata from https://rubygems.org/..........
   Resolving dependencies...
   Using bundler 2.4.13
   Fetching tzinfo 2.0.6
   Installing tzinfo 2.0.6
   ...
   ```

   > **Note**
   >
   > Untuk menginstal permata secara global lagi nanti, jalankan perintah berikut.
   >
   > ```
   > bundle config set --local system 'true'
   > ```

4. Buat arsip file.zip yang berisi `lambda_function.rb` file dengan kode handler fungsi Anda dan dependensi yang Anda instal pada langkah sebelumnya.

   ```
   zip -r my_deployment_package.zip lambda_function.rb vendor
   ```

Anda akan melihat output seperti yang berikut ini.

```
adding: lambda_function.rb (deflated 37%)
  adding: vendor/ (stored 0%)
  adding: vendor/bundle/ (stored 0%)
  adding: vendor/bundle/ruby/ (stored 0%)
  adding: vendor/bundle/ruby/3.2.0/ (stored 0%)
  adding: vendor/bundle/ruby/3.2.0/build_info/ (stored 0%)
  adding: vendor/bundle/ruby/3.2.0/cache/ (stored 0%)
  adding: vendor/bundle/ruby/3.2.0/cache/aws-eventstream-1.0.1.gem (deflated 36%)
...
```

Membuat layer Ruby untuk dependensi Anda

Untuk mempelajari cara mengemas dependensi Ruby Anda ke dalam lapisan Lambda, lihat. the section called “Lapisan”

Membuat paket penerapan.zip dengan pustaka asli

Banyak permata Ruby umum seperti`nokogiri`,`nio4r`, dan `mysql` berisi ekstensi asli yang ditulis dalam C. Ketika Anda menambahkan pustaka yang berisi kode C ke paket penyebaran Anda, Anda harus membangun paket Anda dengan benar untuk memastikan bahwa itu kompatibel dengan lingkungan eksekusi Lambda.

Untuk aplikasi produksi, kami sarankan untuk membangun dan menerapkan kode Anda menggunakan AWS Serverless Application Model (AWS SAM). AWS SAM Gunakan `sam build --use-container` opsi untuk membangun fungsi Anda di dalam wadah Lambda-like Docker. Untuk mempelajari lebih lanjut AWS SAM cara menggunakan kode fungsi, lihat Membangun aplikasi di Panduan AWS SAM Pengembang.

Untuk membuat paket penerapan.zip yang berisi permata dengan ekstensi asli tanpa menggunakan AWS SAM, Anda dapat menggunakan container untuk menggabungkan dependensi Anda di lingkungan yang sama dengan lingkungan runtime Lambda Ruby. Untuk menyelesaikan langkah-langkah ini, Anda harus menginstal Docker di mesin build Anda. Untuk mempelajari lebih lanjut tentang menginstal Docker, lihat Menginstal Docker Engine.

Untuk membuat paket penyebaran.zip dalam wadah Docker

1. Buat folder di mesin build lokal Anda untuk menyimpan wadah Anda. Di dalam folder itu, buat file bernama `dockerfile` dan tempel kode berikut ke dalamnya.

```
FROM public.ecr.aws/sam/build-ruby3.2:latest-x86_64
RUN gem update bundler
CMD "/bin/bash"
```

2. Di dalam folder yang Anda `dockerfile` buat, jalankan perintah berikut untuk membuat wadah Docker.

```
docker build -t awsruby32 .
```

3. Arahkan ke direktori proyek yang berisi `.rb` file dengan kode handler fungsi Anda dan `Gemfile` menentukan dependensi fungsi Anda. Dari dalam direktori itu, jalankan perintah berikut untuk memulai wadah Lambda Ruby.

   Linux/MacOS

```
docker run --rm -it -v $PWD:/var/task -w /var/task awsruby32
```

   > **Note**
   >
   > Di macOS, Anda mungkin melihat peringatan yang memberi tahu Anda bahwa platform gambar yang diminta tidak cocok dengan platform host yang terdeteksi. Abaikan peringatan ini.

   Windows PowerShell

```
docker run --rm -it -v ${pwd}:/var/task -w /var/task awsruby32
```

   Saat wadah Anda dimulai, Anda akan melihat prompt bash.

```
bash-4.2#
```

4. Konfigurasikan utilitas bundel untuk menginstal permata yang ditentukan `Gemfile` dalam `vendor/bundle` direktori lokal Anda dan instal dependensi Anda.

   ```
   bash-4.2# bundle config set --local path 'vendor/bundle' && bundle install
   ```

5. Buat paket deployment .zip dengan kode fungsi Anda dan dependensinya. Dalam contoh ini, file yang berisi kode handler fungsi Anda diberi nama`lambda_function.rb`.

   ```
   bash-4.2# zip -r my_deployment_package.zip lambda_function.rb vendor
   ```

6. Keluar dari wadah dan kembali ke direktori proyek lokal Anda.

   ```
   bash-4.2# exit
   ```

   Anda sekarang dapat menggunakan paket penyebaran file.zip untuk membuat atau memperbarui fungsi Lambda Anda. Lihat Membuat dan memperbarui Fungsi Acara Ruby Lambda menggunakan file.zip

Membuat dan memperbarui Fungsi Acara Ruby Lambda menggunakan file.zip

Setelah Anda membuat paket.zip deployment, Anda dapat menggunakannya untuk membuat fungsi Lambda baru atau memperbarui yang sudah ada. Anda dapat menerapkan paket.zip Anda menggunakan konsol Lambda, API, AWS Command Line Interface dan Lambda. Anda juga dapat membuat dan memperbarui fungsi Lambda menggunakan AWS Serverless Application Model (AWS SAM) dan. CloudFormation

Ukuran maksimum untuk paket.zip deployment untuk Lambda adalah 250 MB (unzip). Perhatikan bahwa batas ini berlaku untuk ukuran gabungan semua file yang Anda unggah, termasuk lapisan Lambda apa pun.

Runtime Lambda membutuhkan izin untuk membaca file dalam paket deployment Anda. Dalam notasi oktal izin Linux, Lambda membutuhkan 644 izin untuk file yang tidak dapat dieksekusi (rw-r - r--) dan 755 izin (rwxr-xr-x) untuk direktori dan file yang dapat dieksekusi.

Di Linux dan macOS, gunakan `chmod` perintah untuk mengubah izin file pada file dan direktori dalam paket penyebaran Anda. Misalnya, untuk memberikan file yang tidak dapat dieksekusi izin yang benar, jalankan perintah berikut.

```
chmod 644 <filepath>
```

Untuk mengubah izin file di Windows, lihat Mengatur, Melihat, Mengubah, atau Menghapus Izin pada Objek dalam dokumentasi Microsoft Windows.

> **Note**
>
> Jika Anda tidak memberikan Lambda izin yang diperlukan untuk mengakses direktori dalam paket penerapan Anda, Lambda menetapkan izin untuk direktori tersebut ke 755 (rwxr-xr-x).

Membuat dan memperbarui fungsi dengan file.zip menggunakan konsol

Untuk membuat fungsi baru, Anda harus terlebih dahulu membuat fungsi di konsol, lalu mengunggah arsip.zip Anda. Untuk memperbarui fungsi yang ada, buka halaman untuk fungsi Anda, lalu ikuti prosedur yang sama untuk menambahkan file.zip Anda yang diperbarui.

Jika file.zip Anda kurang dari 50MB, Anda dapat membuat atau memperbarui fungsi dengan mengunggah file langsung dari mesin lokal Anda. Untuk file.zip yang lebih besar dari 50MB, Anda harus mengunggah paket Anda ke bucket Amazon S3 terlebih dahulu. Untuk petunjuk tentang cara mengunggah file ke bucket Amazon S3 menggunakan Konsol Manajemen AWS, lihat Memulai Amazon S3. Untuk mengunggah file menggunakan AWS CLI, lihat Memindahkan objek di Panduan AWS CLI Pengguna.

> **Note**
>
> Anda tidak dapat mengubah jenis paket penerapan (.zip atau image kontainer) untuk fungsi yang ada. Misalnya, Anda tidak dapat mengonversi fungsi gambar kontainer untuk menggunakan arsip file.zip. Anda harus membuat fungsi baru.

Untuk membuat fungsi baru (konsol)

1. Buka halaman Fungsi Acara konsol Lambda dan pilih Buat Fungsi Acara.
2. Pilih Tulis dari awal.
3. Di bagian Informasi dasar, lakukan hal berikut:
   a. Untuk nama Fungsi, masukkan nama untuk fungsi Anda.
   b. Untuk Runtime, pilih runtime yang ingin Anda gunakan.

c. (Opsional) Untuk Arsitektur, pilih arsitektur set instruksi untuk fungsi Anda. Arsitektur defaultnya adalah x86_64. Pastikan bahwa paket deployment .zip untuk fungsi Anda kompatibel dengan arsitektur set instruksi yang Anda pilih.

4. (Opsional) Di bagian Izin, luaskan Ubah peran eksekusi default. Anda dapat membuat peran Eksekusi baru atau menggunakan yang sudah ada.
5. Pilih Buat Fungsi Acara. Lambda menciptakan fungsi dasar 'Hello world' menggunakan runtime yang Anda pilih.

Untuk mengunggah arsip.zip dari mesin lokal Anda (konsol)

1. Di halaman Fungsi Acara konsol Lambda, pilih fungsi yang ingin Anda unggah file.zip.
2. Pilih tab Kode.
3. Di panel Sumber kode, pilih Unggah dari.
4. Pilih file.zip.
5. Untuk mengunggah file.zip, lakukan hal berikut:

   a. Pilih Unggah, lalu pilih file.zip Anda di pemilih file.

   b. Pilih Buka.

   c. Pilih Simpan.

Untuk mengunggah arsip.zip dari bucket Amazon S3 (konsol)

1. Di halaman Fungsi Acara konsol Lambda, pilih fungsi yang ingin Anda unggah file.zip baru.
2. Pilih tab Kode.
3. Di panel Sumber kode, pilih Unggah dari.
4. Pilih lokasi Amazon S3.
5. Rekatkan URL tautan Amazon S3 dari file.zip Anda dan pilih Simpan.

Memperbarui fungsi file.zip menggunakan editor kode konsol

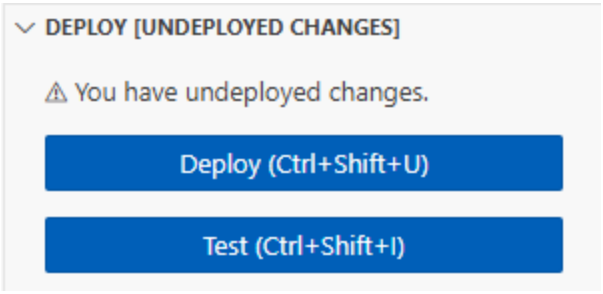
Untuk beberapa fungsi dengan paket penyebaran.zip, Anda dapat menggunakan editor kode bawaan konsol Lambda untuk memperbarui kode fungsi Anda secara langsung. Untuk menggunakan fitur ini, fungsi Anda harus memenuhi kriteria berikut:

- Fungsi Anda harus menggunakan salah satu runtime bahasa yang ditafsirkan (Python,, Node.js atau Ruby)
- Paket penerapan fungsi Anda harus lebih kecil dari 50 MB (unzip).

Kode fungsi untuk fungsi dengan paket penerapan gambar kontainer tidak dapat diedit langsung di konsol.

Untuk memperbarui kode fungsi menggunakan editor kode konsol

1. Buka halaman Fungsi Acara konsol Lambda dan pilih fungsi Anda.
2. Pilih tab Kode.
3. Di panel Sumber kode, pilih file kode sumber Anda dan edit di editor kode terintegrasi.
4. Di bagian DEPLOY, pilih Deploy untuk memperbarui kode fungsi Anda:



Membuat dan memperbarui fungsi dengan file.zip menggunakan AWS CLI

Anda dapat menggunakan AWS CLIuntuk membuat fungsi baru atau memperbarui yang sudah ada menggunakan file.zip. Gunakan perintah create-function dan update-function-code untuk menyebarkan paket.zip Anda. Jika file.zip Anda lebih kecil dari 50MB, Anda dapat mengunggah paket.zip dari lokasi file di mesin build lokal Anda. Untuk file yang lebih besar, Anda harus mengunggah paket.zip Anda dari bucket Amazon S3. Untuk petunjuk tentang cara mengunggah file ke bucket Amazon S3 menggunakan AWS CLI, lihat Memindahkan objek di AWS CLI Panduan Pengguna.

> **Note**
>
> Jika Anda mengunggah file.zip dari bucket Amazon S3 menggunakan AWS CLI bucket, bucket harus berada di lokasi Wilayah AWS yang sama dengan fungsi Anda.

Untuk membuat fungsi baru menggunakan file.zip dengan AWS CLI, Anda harus menentukan yang berikut:

- Nama fungsi Anda (`--function-name`)
- Runtime () `--runtime` fungsi Anda
- Nama Sumber Daya Amazon (ARN) dari peran eksekusi fungsi Anda () `--role`
- Nama metode handler dalam kode fungsi Anda () `--handler`

Anda juga harus menentukan lokasi file.zip Anda. Jika file.zip Anda terletak di folder di mesin build lokal Anda, gunakan `--zip-file` opsi untuk menentukan jalur file, seperti yang ditunjukkan pada perintah contoh berikut.

```
aws lambda create-function --function-name myFunction \
--runtime ruby3.2 --handler lambda_function.lambda_handler \
--role arn:aws:iam::111122223333:role/service-role/my-lambda-role \
--zip-file fileb://myFunction.zip
```

Untuk menentukan lokasi file.zip di bucket Amazon S3, gunakan opsi seperti `--code` yang ditunjukkan pada perintah contoh berikut. Anda hanya perlu menggunakan `S3ObjectVersion` parameter untuk objek berversi.

```
aws lambda create-function --function-name myFunction \
--runtime ruby3.2 --handler lambda_function.lambda_handler \
--role arn:aws:iam::111122223333:role/service-role/my-lambda-role \
--code S3Bucket=amzn-s3-demo-
bucket,S3Key=myFileName.zip,S3ObjectVersion=myObjectVersion
```

Untuk memperbarui fungsi yang ada menggunakan CLI, Anda menentukan nama fungsi Anda menggunakan parameter. `--function-name` Anda juga harus menentukan lokasi file.zip yang ingin Anda gunakan untuk memperbarui kode fungsi Anda. Jika file.zip Anda terletak di folder di mesin build lokal Anda, gunakan `--zip-file` opsi untuk menentukan jalur file, seperti yang ditunjukkan pada perintah contoh berikut.

```
aws lambda update-function-code --function-name myFunction \
--zip-file fileb://myFunction.zip
```

Untuk menentukan lokasi file.zip di bucket Amazon S3, gunakan opsi `--s3-key` dan seperti `--s3-bucket` yang ditunjukkan pada perintah contoh berikut. Anda hanya perlu menggunakan `--s3-object-version` parameter untuk objek berversi.

```
aws lambda update-function-code --function-name myFunction \
--s3-bucket amzn-s3-demo-bucket --s3-key myFileName.zip --s3-object-version myObject
 Version
```

Membuat dan memperbarui fungsi dengan file.zip menggunakan API Lambda

Untuk membuat dan memperbarui fungsi menggunakan arsip file.zip, gunakan operasi API berikut:

- CreateFunction
- UpdateFunctionCode

Membuat dan memperbarui fungsi dengan file.zip menggunakan AWS SAM

The AWS Serverless Application Model (AWS SAM) adalah toolkit yang membantu merampingkan proses membangun dan menjalankan aplikasi tanpa server. AWS Anda menentukan sumber daya untuk aplikasi Anda dalam template YAMB atau JSON dan menggunakan antarmuka baris AWS SAM perintah (AWS SAM CLI) untuk membangun, mengemas, dan menyebarkan aplikasi Anda. Saat Anda membuat fungsi Lambda dari AWS SAM template, AWS SAM secara otomatis membuat paket penerapan .zip atau gambar kontainer dengan kode fungsi Anda dan dependensi apa pun yang Anda tentukan. Untuk mempelajari lebih lanjut cara menggunakan AWS SAM untuk membangun dan menerapkan fungsi Lambda, lihat Memulai di Panduan AWS SAMAWS Serverless Application Model Pengembang.

Anda juga dapat menggunakan AWS SAM untuk membuat fungsi Lambda menggunakan arsip file.zip yang ada. Untuk membuat fungsi Lambda menggunakan AWS SAM, Anda dapat menyimpan file.zip di bucket Amazon S3 atau di folder lokal di mesin build Anda. Untuk petunjuk tentang cara mengunggah file ke bucket Amazon S3 menggunakan AWS CLI, lihat Memindahkan objek di AWS CLI Panduan Pengguna.

Dalam AWS SAM template Anda, `AWS::Serverless::Function` sumber daya menentukan fungsi Lambda Anda. Dalam sumber daya ini, atur properti berikut untuk membuat fungsi menggunakan arsip file.zip:

- `PackageType`- diatur ke `Zip`
- `CodeUri`- diatur ke kode fungsi Amazon S3 URI, jalur ke folder lokal, atau objek FunctionCode
- `Runtime`- Setel ke runtime yang Anda pilih

Dengan AWS SAM, jika file.zip Anda lebih besar dari 50MB, Anda tidak perlu mengunggahnya ke bucket Amazon S3 terlebih dahulu. AWS SAM dapat mengunggah paket.zip hingga ukuran maksimum yang diizinkan 250MB (unzip) dari lokasi di mesin build lokal Anda.

Untuk mempelajari selengkapnya tentang penerapan fungsi menggunakan file.zip di AWS SAM, lihat AWS::Serverless: :Fungsi dalam Panduan PengembangAWS SAM .

Membuat dan memperbarui fungsi dengan file.zip menggunakan CloudFormation

Anda dapat menggunakan CloudFormation untuk membuat fungsi Lambda menggunakan arsip file.zip. Untuk membuat fungsi Lambda dari file.zip, Anda harus terlebih dahulu mengunggah file Anda ke bucket Amazon S3. Untuk petunjuk tentang cara mengunggah file ke bucket Amazon S3 menggunakan AWS CLI, lihat Memindahkan objek di AWS CLI Panduan Pengguna.

Dalam CloudFormation template Anda, `AWS::Lambda::Function` sumber daya menentukan fungsi Lambda Anda. Dalam sumber daya ini, atur properti berikut untuk membuat fungsi menggunakan arsip file.zip:

- `PackageType`- Setel ke `Zip`
- `Code`- Masukkan nama bucket Amazon S3 dan nama file.zip di dan bidang `S3Bucket` `S3Key`
- `Runtime`- Setel ke runtime yang Anda pilih

File.zip yang CloudFormation menghasilkan tidak boleh melebihi 4MB. Untuk mempelajari selengkapnya tentang penerapan fungsi menggunakan file.zip di CloudFormation, lihat AWS::Lambda: :Fungsi dalam Panduan Pengguna.CloudFormation

Menyebarkan Fungsi Acara Ruby Lambda dengan gambar kontainer

Ada tiga cara untuk membangun image container untuk Ruby Lambda Event Function:

- Menggunakan sebuah AWS gambar dasar untuk Ruby

  Gambar AWS dasar dimuat sebelumnya dengan runtime bahasa, klien antarmuka runtime untuk mengelola interaksi antara Lambda dan kode fungsi Anda, dan emulator antarmuka runtime untuk pengujian lokal.

- Menggunakan sebuah AWS OS-only gambar dasar

  AWS OS-only image dasar berisi distribusi Amazon Linux dan emulator antarmuka runtime. Gambar-gambar ini biasanya digunakan untuk membuat gambar kontainer untuk bahasa yang dikompilasi, seperti Go dan Rust, dan untuk versi bahasa atau bahasa yang Lambda tidak menyediakan gambar dasar, seperti Node.js 19. Anda juga dapat menggunakan gambar OS-only dasar untuk menerapkan runtime kustom. Untuk membuat gambar kompatibel dengan Lambda, Anda harus menyertakan klien antarmuka runtime untuk Ruby dalam gambar.

- Menggunakan gambar AWS non-dasar

  Anda dapat menggunakan gambar dasar alternatif dari registri kontainer lain, seperti Alpine Linux atau Debian. Anda juga dapat menggunakan gambar kustom yang dibuat oleh organisasi Anda. Untuk membuat gambar kompatibel dengan Lambda, Anda harus menyertakan klien antarmuka runtime untuk Ruby dalam gambar.

> **Tip**
>
> Untuk mengurangi waktu yang dibutuhkan agar fungsi kontainer Lambda menjadi aktif, lihat Menggunakan build multi-tahap dalam dokumentasi Docker. Untuk membuat gambar kontainer yang efisien, ikuti Praktik terbaik untuk menulis Dockerfiles.

Halaman ini menjelaskan cara membuat, menguji, dan menyebarkan gambar kontainer untuk Lambda.

Topik

- AWS gambar dasar untuk Ruby
- Menggunakan sebuah AWS gambar dasar untuk Ruby

- Menggunakan gambar dasar alternatif dengan klien antarmuka runtime

AWS gambar dasar untuk Ruby

AWS menyediakan gambar dasar berikut untuk Ruby:

Tanda	Waktu berjalan	Sistem operasi	Dockerfile	penghentian
4.0	Ruby 4.0	Amazon Linux 2023	Dockerfile untuk Ruby 4.0 di GitHub	31 Mar 2029
3.4	Ruby 3.4	Amazon Linux 2023	Dockerfile untuk Ruby 3.4 di GitHub	31 Mar 2028
3.3	Ruby 3.3	Amazon Linux 2023	Dockerfile untuk Ruby 3.3 di GitHub	31 Mar 2027

Repositori Amazon ECR: gallery.ecr. aws/lambda/ruby

Menggunakan sebuah AWS gambar dasar untuk Ruby

Prasyarat

Untuk menyelesaikan langkah-langkah di bagian ini, Anda harus memiliki hal-hal berikut:

- AWS CLI versi 2
- Docker (versi minimum 25.0.0)
- Plugin buildx Docker.
- Ruby

Membuat gambar dari gambar dasar

Untuk membuat gambar kontainer untuk Ruby

1. Buat direktori untuk proyek, dan kemudian beralih ke direktori itu.

```
mkdir example
```

```
cd example
```

2. Buat file baru bernama`Gemfile`. Di sinilah Anda mencantumkan RubyGems paket yang diperlukan aplikasi Anda. AWS SDK untuk Ruby Tersedia dari RubyGems. Anda harus memilih permata AWS layanan tertentu untuk dipasang. Misalnya, untuk menggunakan permata Ruby untuk Lambda, Gemfile Anda akan terlihat seperti ini:

```
source 'https://rubygems.org'

gem 'aws-sdk-lambda'
```

   Atau, permata aws-sdk berisi setiap permata layanan yang tersedia. AWS Permata ini sangat besar. Kami menyarankan Anda menggunakannya hanya jika Anda bergantung pada banyak AWS layanan.

3. Instal dependensi yang ditentukan dalam Gemfile menggunakan bundle install.

```
bundle install
```

4. Buat file baru bernama`lambda_function.rb`. Anda dapat menambahkan kode fungsi contoh berikut ke file untuk pengujian, atau menggunakan kode Anda sendiri.

   Example Fungsi Ruby

```
module LambdaFunction
  class Handler
    def self.process(event:,context:)
      "Hello from Lambda!"
    end
  end
end
```

5. Buat Dockerfile baru. Berikut ini adalah contoh Dockerfile yang menggunakan gambar AWS dasar. Dockerfiles ini menggunakan konfigurasi berikut:

   - Atur properti `FROM` untuk URI dari gambar dasar.
   - Gunakan perintah COPY untuk menyalin kode fungsi dan dependensi runtime ke`{LAMBDA_TASK_ROOT}`, variabel lingkungan. Lambda-defined
   - Atur `CMD` argumen ke penangan fungsi Lambda.

Perhatikan bahwa contoh Dockerfile tidak menyertakan instruksi USER. Saat Anda menyebarkan gambar kontainer ke Lambda, Lambda secara otomatis mendefinisikan pengguna Linux default dengan izin yang paling tidak memiliki hak istimewa. Ini berbeda dari perilaku Docker standar yang default ke `root` pengguna ketika tidak ada USER instruksi yang diberikan.

Example Dockerfile

```
FROM public.ecr.aws/lambda/ruby:3.4

# Copy Gemfile and Gemfile.lock
COPY Gemfile Gemfile.lock ${LAMBDA_TASK_ROOT}/

# Install Bundler and the specified gems
RUN gem install bundler:2.4.20 && \
    bundle config set --local path 'vendor/bundle' && \
    bundle install

# Copy function code
COPY lambda_function.rb ${LAMBDA_TASK_ROOT}/

# Set the CMD to your handler (could also be done as a parameter override outside
 of the Dockerfile)
CMD [ "lambda_function.LambdaFunction::Handler.process" ]
```

6. Buat image Docker dengan perintah docker build. Contoh berikut menamai gambar `docker-image` dan memberinya `test` tag. Untuk membuat gambar Anda kompatibel dengan Lambda, Anda harus menggunakan opsi. `--provenance=false`

```
docker buildx build --platform linux/amd64 --provenance=false -t docker-image:test
 .
```

> **Note**
>
> Perintah menentukan `--platform linux/amd64` opsi untuk memastikan bahwa container Anda kompatibel dengan lingkungan eksekusi Lambda terlepas dari arsitektur mesin build Anda. Jika Anda berniat untuk membuat fungsi Lambda menggunakan arsitektur set instruksi ARM64, pastikan untuk mengubah perintah untuk menggunakan opsi sebagai `--platform linux/arm64` gantinya.

(Opsional) Uji gambar secara lokal

1. Mulai gambar Docker dengan perintah docker run. Dalam contoh ini, `docker-image` adalah nama gambar dan `test` tag.

   ```
   docker run --platform linux/amd64 -p 9000:8080 docker-image:test
   ```

   Perintah ini menjalankan gambar sebagai wadah dan membuat titik akhir lokal di`localhost:9000/2015-03-31/functions/function/invocations`.

   > **Note**
   >
   > Jika Anda membuat image Docker untuk arsitektur set instruksi ARM64, pastikan untuk menggunakan `--platform linux/arm64` opsi sebagai gantinya. `--platform linux/amd64`

2. Dari jendela terminal baru, posting acara ke titik akhir lokal.

   Linux/macOS

   Di Linux dan macOS, jalankan perintah berikut: `curl`

   ```
   curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
   ```

   Perintah ini memanggil fungsi dengan peristiwa kosong dan mengembalikan respons. Jika Anda menggunakan kode fungsi Anda sendiri daripada kode fungsi sampel, Anda mungkin ingin memanggil fungsi dengan payload JSON. Contoh:

   ```
   curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d
    '{"payload":"hello world!"}'
   ```

   PowerShell

   Dalam PowerShell, jalankan `Invoke-WebRequest` perintah berikut:

   ```
   Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
   invocations" -Method Post -Body '{}' -ContentType "application/json"
   ```

Perintah ini memanggil fungsi dengan peristiwa kosong dan mengembalikan respons. Jika Anda menggunakan kode fungsi Anda sendiri daripada kode fungsi sampel, Anda mungkin ingin memanggil fungsi dengan payload JSON. Contoh:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
invocations" -Method Post -Body '{"payload":"hello world!"}' -ContentType
 "application/json"
```

3. Dapatkan ID kontainer.

```
docker ps
```

4. Gunakan perintah docker kill untuk menghentikan wadah. Dalam perintah ini, ganti `3766c4ab331c` dengan ID kontainer dari langkah sebelumnya.

```
docker kill 3766c4ab331c
```

Menyebarkan gambar

Untuk mengunggah gambar ke Amazon ECR dan membuat fungsi Lambda

1. Jalankan perintah get-login-password untuk mengautentikasi CLI Docker ke registri Amazon ECR Anda.
   - Tetapkan `--region` nilai ke Wilayah AWS tempat Anda ingin membuat repositori Amazon ECR.
   - Ganti `111122223333` dengan Akun AWS ID Anda.

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --
password-stdin 111122223333.dkr.ecr.us-east-1.amazonaws.com
```

2. Buat repositori di Amazon ECR menggunakan perintah create-repository.

```
aws ecr create-repository --repository-name hello-world --region us-east-1 --image-
scanning-configuration scanOnPush=true --image-tag-mutability MUTABLE
```

> **Note**
>
> Repositori Amazon ECR harus sama Wilayah AWS dengan fungsi Lambda.

Jika berhasil, Anda melihat respons seperti ini:

```
{
    "repository": {
        "repositoryArn": "arn:aws:ecr:us-east-1:111122223333:repository/hello-
world",
        "registryId": "111122223333",
        "repositoryName": "hello-world",
        "repositoryUri": "111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-
world",
        "createdAt": "2023-03-09T10:39:01+00:00",
        "imageTagMutability": "MUTABLE",
        "imageScanningConfiguration": {
            "scanOnPush": true
        },
        "encryptionConfiguration": {
            "encryptionType": "AES256"
        }
    }
}
```

3. Salin `repositoryUri` dari output pada langkah sebelumnya.
4. Jalankan perintah tag docker untuk menandai gambar lokal Anda ke repositori Amazon ECR Anda sebagai versi terbaru. Dalam perintah ini:

   - `docker-image:test`adalah nama dan tag gambar Docker Anda. Ini adalah nama gambar dan tag yang Anda tentukan dalam `docker build` perintah.
   - Ganti <ECRrepositoryUri> dengan `repositoryUri` yang Anda salin. Pastikan untuk menyertakan `:latest` di akhir URI.

```
docker tag docker-image:test <ECRrepositoryUri>:latest
```

Contoh:

```
docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

5. Jalankan perintah docker push untuk menyebarkan gambar lokal Anda ke repositori Amazon ECR. Pastikan untuk menyertakan `:latest` di akhir URI repositori.

```
docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

6. Buat peran eksekusi untuk fungsi tersebut, jika Anda belum memilikinya. Anda memerlukan Nama Sumber Daya Amazon (ARN) dari peran tersebut di langkah berikutnya.
7. Buat fungsi Lambda. Untuk`ImageUri`, tentukan URI repositori dari sebelumnya. Pastikan untuk menyertakan `:latest` di akhir URI.

```
aws lambda create-function \
  --function-name hello-world \
  --package-type Image \
  --code ImageUri=111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
  --role arn:aws:iam::111122223333:role/lambda-ex
```

> **Note**
>
> Anda dapat membuat fungsi menggunakan gambar di AWS akun yang berbeda, selama gambar berada di Wilayah yang sama dengan fungsi Lambda. Untuk informasi selengkapnya, lihat Izin lintas akun Amazon ECR.

8. Memanggil fungsi.

```
aws lambda invoke --function-name hello-world response.json
```

Anda akan melihat tanggapan seperti ini:

```
{
  "ExecutedVersion": "$LATEST",
  "StatusCode": 200
}
```

9. Untuk melihat output dari fungsi, periksa `response.json` file.

Untuk memperbarui kode fungsi, Anda harus membuat gambar lagi, mengunggah gambar baru ke repositori Amazon ECR, dan kemudian menggunakan perintah update-function-code untuk menyebarkan gambar ke fungsi Lambda.

Lambda menyelesaikan tag gambar ke intisari gambar tertentu. Ini berarti bahwa jika Anda mengarahkan tag gambar yang digunakan untuk menyebarkan fungsi ke gambar baru di Amazon ECR, Lambda tidak secara otomatis memperbarui fungsi untuk menggunakan gambar baru.

Untuk menyebarkan gambar baru ke fungsi Lambda yang sama, Anda harus menggunakan perintah update-function-code, meskipun tag gambar di Amazon ECR tetap sama. Dalam contoh berikut, `--publish` opsi membuat versi baru dari fungsi menggunakan gambar kontainer yang diperbarui.

```
aws lambda update-function-code \
  --function-name hello-world \
  --image-uri 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
  --publish
```

Menggunakan gambar dasar alternatif dengan klien antarmuka runtime

Jika Anda menggunakan gambar OS-only dasar atau gambar dasar alternatif, Anda harus menyertakan klien antarmuka runtime dalam gambar Anda. Klien antarmuka runtime memperluasAPI runtime, yang mengelola interaksi antara Lambda dan kode fungsi Anda.

Instal klien antarmuka runtime Lambda untuk Ruby menggunakan manajer paket: RubyGems.org

```
gem install aws_lambda_ric
```

Anda juga dapat mengunduh klien antarmuka runtime Ruby dari. GitHub

Contoh berikut menunjukkan bagaimana membangun image container untuk Ruby menggunakan gambar AWS non-dasar. Contoh Dockerfile menggunakan gambar dasar Ruby resmi. Dockerfile menyertakan klien antarmuka runtime.

Prasyarat

Untuk menyelesaikan langkah-langkah di bagian ini, Anda harus memiliki hal-hal berikut:

- AWS CLI versi 2
- Docker (versi minimum 25.0.0)

- Plugin buildx Docker.
- Ruby

Membuat gambar dari gambar dasar alternatif

Untuk membuat gambar kontainer untuk Ruby menggunakan gambar dasar alternatif

1. Buat direktori untuk proyek, dan kemudian beralih ke direktori itu.

```
mkdir example
cd example
```

2. Buat file baru bernama`Gemfile`. Di sinilah Anda mencantumkan RubyGems paket yang diperlukan aplikasi Anda. AWS SDK untuk Ruby Tersedia dari RubyGems. Anda harus memilih permata AWS layanan tertentu untuk dipasang. Misalnya, untuk menggunakan permata Ruby untuk Lambda, Gemfile Anda akan terlihat seperti ini:

```
source 'https://rubygems.org'

gem 'aws-sdk-lambda'
```

   Atau, permata aws-sdk berisi setiap permata layanan yang tersedia. AWS Permata ini sangat besar. Kami menyarankan Anda menggunakannya hanya jika Anda bergantung pada banyak AWS layanan.

3. Instal dependensi yang ditentukan dalam Gemfile menggunakan bundle install.

```
bundle install
```

4. Buat file baru bernama`lambda_function.rb`. Anda dapat menambahkan kode fungsi contoh berikut ke file untuk pengujian, atau menggunakan kode Anda sendiri.

   Example Fungsi Ruby

```
module LambdaFunction
  class Handler
    def self.process(event:,context:)
      "Hello from Lambda!"
    end
  end
```

```
end
```

5. Buat Dockerfile baru. Dockerfile berikut menggunakan gambar dasar Ruby alih-alih gambar dasar.AWS Dockerfile menyertakan klien antarmuka runtime untuk Ruby, yang membuat gambar kompatibel dengan Lambda. Atau, Anda dapat menambahkan klien antarmuka runtime ke Gemfile aplikasi Anda.

   - Atur `FROM` properti ke gambar dasar Ruby.
   - Buat direktori untuk kode fungsi dan variabel lingkungan yang menunjuk ke direktori itu. Dalam contoh ini, direktorinya adalah`/var/task`, yang mencerminkan lingkungan eksekusi Lambda. Namun, Anda dapat memilih direktori apa pun untuk kode fungsi karena Dockerfile tidak menggunakan gambar AWS dasar.
   - Atur `ENTRYPOINT` ke modul yang Anda inginkan untuk menjalankan wadah Docker saat dimulai. Dalam hal ini, modul adalah klien antarmuka runtime.
   - Atur `CMD` argumen ke penangan fungsi Lambda.

   Perhatikan bahwa contoh Dockerfile tidak menyertakan instruksi USER. Saat Anda menyebarkan gambar kontainer ke Lambda, Lambda secara otomatis mendefinisikan pengguna Linux default dengan izin yang paling tidak memiliki hak istimewa. Ini berbeda dari perilaku Docker standar yang default ke `root` pengguna ketika tidak ada `USER` instruksi yang diberikan.

   Example Dockerfile

```
FROM ruby:2.7

# Install the runtime interface client for Ruby
RUN gem install aws_lambda_ric

# Add the runtime interface client to the PATH
ENV PATH="/usr/local/bundle/bin:${PATH}"

# Create a directory for the Lambda function
ENV LAMBDA_TASK_ROOT=/var/task
RUN mkdir -p ${LAMBDA_TASK_ROOT}
WORKDIR ${LAMBDA_TASK_ROOT}

# Copy Gemfile and Gemfile.lock
COPY Gemfile Gemfile.lock ${LAMBDA_TASK_ROOT}/
```

```
# Install Bundler and the specified gems
RUN gem install bundler:2.4.20 && \
    bundle config set --local path 'vendor/bundle' && \
    bundle install

# Copy function code
COPY lambda_function.rb ${LAMBDA_TASK_ROOT}/

# Set runtime interface client as default command for the container runtime
ENTRYPOINT [ "aws_lambda_ric" ]

# Set the CMD to your handler (could also be done as a parameter override outside
 of the Dockerfile)
CMD [ "lambda_function.LambdaFunction::Handler.process" ]
```

6. Buat image Docker dengan perintah docker build. Contoh berikut menamai gambar `docker-image` dan memberinya `test` tag. Untuk membuat gambar Anda kompatibel dengan Lambda, Anda harus menggunakan opsi. `--provenance=false`

```
docker buildx build --platform linux/amd64 --provenance=false -t docker-image:test
 .
```

> **Note**
>
> Perintah menentukan `--platform linux/amd64` opsi untuk memastikan bahwa container Anda kompatibel dengan lingkungan eksekusi Lambda terlepas dari arsitektur mesin build Anda. Jika Anda berniat untuk membuat fungsi Lambda menggunakan arsitektur set instruksi ARM64, pastikan untuk mengubah perintah untuk menggunakan opsi sebagai `--platform linux/arm64` gantinya.

(Opsional) Uji gambar secara lokal

Gunakan emulator antarmuka runtime untuk menguji gambar secara lokal. Anda dapat membangun emulator ke dalam gambar Anda atau menggunakan prosedur berikut untuk menginstalnya di mesin lokal Anda.

Untuk menginstal dan menjalankan emulator antarmuka runtime di mesin lokal Anda

1. Dari direktori proyek Anda, jalankan perintah berikut untuk mengunduh emulator antarmuka runtime (arsitektur x86-64) dari GitHub dan menginstalnya di mesin lokal Anda.

   Linux/macOS

   ```
   mkdir -p ~/.aws-lambda-rie && \
       curl -Lo ~/.aws-lambda-rie/aws-lambda-rie https://github.com/aws/aws-lambda-
   runtime-interface-emulator/releases/latest/download/aws-lambda-rie && \
       chmod +x ~/.aws-lambda-rie/aws-lambda-rie
   ```

   Untuk menginstal emulator arm64, ganti URL GitHub repositori di perintah sebelumnya dengan yang berikut:

   ```
   https://github.com/aws/aws-lambda-runtime-interface-emulator/releases/latest/
   download/aws-lambda-rie-arm64
   ```

   PowerShell

   ```
   $dirPath = "$HOME\.aws-lambda-rie"
   if (-not (Test-Path $dirPath)) {
       New-Item -Path $dirPath -ItemType Directory
   }

   $downloadLink = "https://github.com/aws/aws-lambda-runtime-interface-emulator/
   releases/latest/download/aws-lambda-rie"
   $destinationPath = "$HOME\.aws-lambda-rie\aws-lambda-rie"
   Invoke-WebRequest -Uri $downloadLink -OutFile $destinationPath
   ```

   Untuk menginstal emulator arm64, ganti `$downloadLink` dengan yang berikut ini:

   ```
   https://github.com/aws/aws-lambda-runtime-interface-emulator/releases/latest/
   download/aws-lambda-rie-arm64
   ```

2. Mulai gambar Docker dengan perintah docker run. Perhatikan hal-hal berikut:

   - `docker-image`adalah nama gambar dan `test` tag.
   - `aws_lambda_ric lambda_function.LambdaFunction::Handler.process`adalah `ENTRYPOINT` diikuti oleh `CMD` dari Dockerfile Anda.

Linux/macOS

```
docker run --platform linux/amd64 -d -v ~/.aws-lambda-rie:/aws-lambda -p
 9000:8080 \
    --entrypoint /aws-lambda/aws-lambda-rie \
    docker-image:test \
        aws_lambda_ric lambda_function.LambdaFunction::Handler.process
```

PowerShell

```
docker run --platform linux/amd64 -d -v "$HOME\.aws-lambda-rie:/aws-lambda" -p
 9000:8080 `
--entrypoint /aws-lambda/aws-lambda-rie `
docker-image:test `
    aws_lambda_ric lambda_function.LambdaFunction::Handler.process
```

Perintah ini menjalankan gambar sebagai wadah dan membuat titik akhir lokal di`localhost:9000/2015-03-31/functions/function/invocations`.

> **Note**
>
> Jika Anda membuat image Docker untuk arsitektur set instruksi ARM64, pastikan untuk menggunakan `--platform linux/`*arm64* opsi sebagai gantinya. `--platform linux/`*amd64*

3. Posting acara ke titik akhir lokal.

Linux/macOS

Di Linux dan macOS, jalankan perintah berikut: `curl`

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
```

Perintah ini memanggil fungsi dengan peristiwa kosong dan mengembalikan respons. Jika Anda menggunakan kode fungsi Anda sendiri daripada kode fungsi sampel, Anda mungkin ingin memanggil fungsi dengan payload JSON. Contoh:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d
 '{"payload":"hello world!"}'
```

PowerShell

Dalam PowerShell, jalankan `Invoke-WebRequest` perintah berikut:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
invocations" -Method Post -Body '{}' -ContentType "application/json"
```

Perintah ini memanggil fungsi dengan peristiwa kosong dan mengembalikan respons. Jika Anda menggunakan kode fungsi Anda sendiri daripada kode fungsi sampel, Anda mungkin ingin memanggil fungsi dengan payload JSON. Contoh:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
invocations" -Method Post -Body '{"payload":"hello world!"}' -ContentType
 "application/json"
```

4. Dapatkan ID kontainer.

```
docker ps
```

5. Gunakan perintah docker kill untuk menghentikan wadah. Dalam perintah ini, ganti `3766c4ab331c` dengan ID kontainer dari langkah sebelumnya.

```
docker kill 3766c4ab331c
```

Menyebarkan gambar

Untuk mengunggah gambar ke Amazon ECR dan membuat fungsi Lambda

1. Jalankan perintah get-login-password untuk mengautentikasi CLI Docker ke registri Amazon ECR Anda.
   - Tetapkan `--region` nilai ke Wilayah AWS tempat Anda ingin membuat repositori Amazon ECR.
   - Ganti `111122223333` dengan Akun AWS ID Anda.

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --
password-stdin 111122223333.dkr.ecr.us-east-1.amazonaws.com
```

2. Buat repositori di Amazon ECR menggunakan perintah create-repository.

```
aws ecr create-repository --repository-name hello-world --region us-east-1 --image-
scanning-configuration scanOnPush=true --image-tag-mutability MUTABLE
```

> **Note**
>
> Repositori Amazon ECR harus sama Wilayah AWS dengan fungsi Lambda.

Jika berhasil, Anda melihat respons seperti ini:

```
{
    "repository": {
        "repositoryArn": "arn:aws:ecr:us-east-1:111122223333:repository/hello-
world",
        "registryId": "111122223333",
        "repositoryName": "hello-world",
        "repositoryUri": "111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-
world",
        "createdAt": "2023-03-09T10:39:01+00:00",
        "imageTagMutability": "MUTABLE",
        "imageScanningConfiguration": {
            "scanOnPush": true
        },
        "encryptionConfiguration": {
            "encryptionType": "AES256"
        }
    }
}
```

3. Salin `repositoryUri` dari output pada langkah sebelumnya.
4. Jalankan perintah tag docker untuk menandai gambar lokal Anda ke repositori Amazon ECR Anda sebagai versi terbaru. Dalam perintah ini:

- `docker-image:test`adalah nama dan tag gambar Docker Anda. Ini adalah nama gambar dan tag yang Anda tentukan dalam `docker build` perintah.
- Ganti `<ECRrepositoryUri>` dengan `repositoryUri` yang Anda salin. Pastikan untuk menyertakan `:latest` di akhir URI.

```
docker tag docker-image:test <ECRrepositoryUri>:latest
```

Contoh:

```
docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

5. Jalankan perintah docker push untuk menyebarkan gambar lokal Anda ke repositori Amazon ECR. Pastikan untuk menyertakan `:latest` di akhir URI repositori.

```
docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

6. Buat peran eksekusi untuk fungsi tersebut, jika Anda belum memilikinya. Anda memerlukan Nama Sumber Daya Amazon (ARN) dari peran tersebut di langkah berikutnya.
7. Buat fungsi Lambda. Untuk`ImageUri`, tentukan URI repositori dari sebelumnya. Pastikan untuk menyertakan `:latest` di akhir URI.

```
aws lambda create-function \
  --function-name hello-world \
  --package-type Image \
  --code ImageUri=111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
  --role arn:aws:iam::111122223333:role/lambda-ex
```

> **Note**
>
> Anda dapat membuat fungsi menggunakan gambar di AWS akun yang berbeda, selama gambar berada di Wilayah yang sama dengan fungsi Lambda. Untuk informasi selengkapnya, lihat Izin lintas akun Amazon ECR.

8. Memanggil fungsi.

```
aws lambda invoke --function-name hello-world response.json
```

Anda akan melihat tanggapan seperti ini:

```
{
  "ExecutedVersion": "$LATEST",
  "StatusCode": 200
}
```

9. Untuk melihat output dari fungsi, periksa `response.json` file.

Untuk memperbarui kode fungsi, Anda harus membuat gambar lagi, mengunggah gambar baru ke repositori Amazon ECR, dan kemudian menggunakan perintah update-function-code untuk menyebarkan gambar ke fungsi Lambda.

Lambda menyelesaikan tag gambar ke intisari gambar tertentu. Ini berarti bahwa jika Anda mengarahkan tag gambar yang digunakan untuk menyebarkan fungsi ke gambar baru di Amazon ECR, Lambda tidak secara otomatis memperbarui fungsi untuk menggunakan gambar baru.

Untuk menyebarkan gambar baru ke fungsi Lambda yang sama, Anda harus menggunakan perintah update-function-code, meskipun tag gambar di Amazon ECR tetap sama. Dalam contoh berikut, `--publish` opsi membuat versi baru dari fungsi menggunakan gambar kontainer yang diperbarui.

```
aws lambda update-function-code \
  --function-name hello-world \
  --image-uri 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
  --publish
```

Bekerja dengan lapisan untuk Fungsi Acara Ruby Lambda

Gunakan lapisan Lambda untuk mengemas kode dan dependensi yang ingin Anda gunakan kembali di beberapa fungsi. Lapisan biasanya berisi dependensi pustaka, runtime kustom, atau file konfigurasi. Membuat layer melibatkan tiga langkah umum:

1. Package konten layer Anda. Ini berarti membuat arsip file.zip yang berisi dependensi yang ingin Anda gunakan dalam fungsi Anda.
2. Buat layer di Lambda.
3. Tambahkan layer ke fungsi Anda.

Topik

- Package konten layer Anda
- Buat layer di Lambda
- Menggunakan permata dari lapisan dalam suatu fungsi
- Tambahkan layer ke fungsi Anda
- Contoh aplikasi

Package konten layer Anda

Untuk membuat layer, bundel paket Anda ke dalam arsip file.zip yang memenuhi persyaratan berikut:

- Buat layer menggunakan versi Ruby yang sama yang Anda rencanakan untuk digunakan untuk fungsi Lambda. Misalnya, jika Anda membuat layer untuk Ruby 4.0, gunakan runtime Ruby 4.0 untuk fungsi Anda.
- File.zip layer Anda harus menggunakan salah satu struktur direktori ini:
  - `ruby/gems/`*`x.x.x`* (di *`x.x.x`* mana versi Ruby Anda, misalnya `3.4.0`)
  - `ruby/lib`

  Untuk informasi selengkapnya, lihat Jalur lapisan untuk setiap runtime Lambda.
- Paket-paket di layer Anda harus kompatibel dengan Linux. Fungsi Lambda berjalan di Amazon Linux.

Anda dapat membuat layer yang berisi permata Ruby pihak ketiga atau modul dan kelas Ruby Anda sendiri. Banyak permata Ruby populer berisi ekstensi asli (kode C) yang harus dikompilasi untuk lingkungan Linux Lambda.

Permata Ruby murni

Permata Ruby murni hanya berisi kode Ruby dan tidak memerlukan kompilasi. Permata ini lebih mudah untuk dikemas dan bekerja di berbagai platform.

Untuk membuat layer menggunakan permata Ruby murni

1. Buat a `Gemfile` untuk menentukan permata Ruby murni yang ingin Anda sertakan dalam layer Anda:

   Example Gemfile

   ```
   source 'https://rubygems.org'

   gem 'tzinfo'
   ```

2. Instal permata ke `vendor/bundle` direktori menggunakan Bundler:

   ```
   bundle config set --local path vendor/bundle
   bundle install
   ```

3. Salin permata yang diinstal ke struktur direktori yang dibutuhkan `ruby/gems/3.4.0` Lambda):

   ```
   mkdir -p ruby/gems/3.4.0
   cp -r vendor/bundle/ruby/3.4.0*/* ruby/gems/3.4.0/
   ```

4. Zip konten lapisan:

   Linux/macOS

   ```
   zip -r layer.zip ruby/
   ```

   PowerShell

   ```
   Compress-Archive -Path .\ruby -DestinationPath .\layer.zip
   ```

Struktur direktori file.zip Anda akan terlihat seperti ini:

```
ruby/
### gems/
    ### 3.4.0/
        ### gems/
        #   ### concurrent-ruby-1.3.5/
        #   ### tzinfo-2.0.6/
        ### specifications/
        ### cache/
        ### build_info/
        ### (other bundler directories)
```

> **Note**
>
> Anda harus meminta setiap permata satu per satu dalam kode fungsi Anda. Anda tidak dapat menggunakan `bundler/setup` atau`Bundler.require`. Untuk informasi selengkapnya, lihat Menggunakan permata dari lapisan dalam suatu fungsi.

Permata dengan ekstensi asli

Banyak permata Ruby populer berisi ekstensi asli (kode C) yang harus dikompilasi untuk platform target. Permata populer dengan ekstensi asli termasuk nokogiri, pg, mysql2, sqlite3, dan ffi. Permata ini harus dibangun di lingkungan Linux yang kompatibel dengan runtime Lambda.

Untuk membuat layer menggunakan permata dengan ekstensi asli

1. Buat a`Gemfile`.

   Example Gemfile

   ```
   source 'https://rubygems.org'

   gem 'nokogiri'
   gem 'httparty'
   ```

2. Gunakan Docker untuk membangun permata di lingkungan Linux yang kompatibel dengan Lambda. Tentukan gambar AWS dasar di Dockerfile Anda:

Example Dockerfile para Ruby 4.0

```
FROM public.ecr.aws/lambda/ruby:3.4

# Copy Gemfile
COPY Gemfile ./

# Install system dependencies for native extensions
RUN dnf update -y && \
    dnf install -y gcc gcc-c++ make

# Configure bundler and install gems
RUN bundle config set --local path vendor/bundle && \
    bundle install

# Create the layer structure
RUN mkdir -p ruby/gems/3.4.0 && \
    cp -r vendor/bundle/ruby/3.4.0*/* ruby/gems/3.4.0/

# Create the layer zip file
RUN zip -r layer.zip ruby/
```

3. Bangun gambar dan ekstrak layernya:

```
docker build -t ruby-layer-builder .
docker run --rm -v $(pwd):/output --entrypoint cp ruby-layer-builder layer.zip /
output/
```

Ini membangun permata di lingkungan Linux yang benar dan menyalin `layer.zip` file ke direktori lokal Anda. Struktur direktori file.zip Anda akan terlihat seperti ini:

```
ruby/
### gems/
    ### 3.4.0/
        ### gems/
        #   ### bigdecimal-3.2.2/
        #   ### csv-3.3.5/
        #   ### httparty-0.23.1/
        #   ### mini_mime-1.1.5/
        #   ### multi_xml-0.7.2/
        #   ### nokogiri-1.18.8-x86_64-linux-gnu/
```

```
        #   ### racc-1.8.1/
        ### build_info/
        ### cache/
        ### specifications/
        ### (other bundler directories)
```

> **Note**
>
> Anda harus meminta setiap permata satu per satu dalam kode fungsi Anda. Anda tidak dapat menggunakan `bundler/setup` atau`Bundler.require`. Untuk informasi selengkapnya, lihat Menggunakan permata dari lapisan dalam suatu fungsi.

Modul Ruby kustom

Untuk membuat layer menggunakan kode Anda sendiri

1. Buat struktur direktori yang diperlukan untuk layer Anda:

   ```
   mkdir -p ruby/lib
   ```

2. Buat modul Ruby Anda di `ruby/lib` direktori. Modul contoh berikut memvalidasi pesanan dengan mengonfirmasi bahwa mereka berisi informasi yang diperlukan.

   Example ruby/lib/order_validator.rb

   ```
   require 'json'

   module OrderValidator
     class ValidationError < StandardError; end

     def self.validate_order(order_data)
       # Validates an order and returns formatted data
       required_fields = %w[product_id quantity]

       # Check required fields
       missing_fields = required_fields.reject { |field| order_data.key?(field) }
       unless missing_fields.empty?
         raise ValidationError, "Missing required fields: #{missing_fields.join(',
    ')}"
       end
   ```

```
    # Validate quantity
    quantity = order_data['quantity']
    unless quantity.is_a?(Integer) && quantity > 0
      raise ValidationError, 'Quantity must be a positive integer'
    end

    # Format and return the validated data
    {
      'product_id' => order_data['product_id'].to_s,
      'quantity' => quantity,
      'shipping_priority' => order_data.fetch('priority', 'standard')
    }
  end

  def self.format_response(status_code, body)
    # Formats the API response
    {
      statusCode: status_code,
      body: JSON.generate(body)
    }
  end
end
```

3. Zip konten lapisan:

   Linux/macOS

   ```
   zip -r layer.zip ruby/
   ```

   PowerShell

   ```
   Compress-Archive -Path .\ruby -DestinationPath .\layer.zip
   ```

   Struktur direktori file.zip Anda akan terlihat seperti ini:

   ```
   ruby/
   ### lib/
       ### order_validator.rb
   ```

4. Dalam fungsi Anda, membutuhkan dan menggunakan modul. Anda harus meminta setiap permata satu per satu dalam kode fungsi Anda. Anda tidak dapat menggunakan `bundler/setup` atau`Bundler.require`. Untuk informasi selengkapnya, lihat Menggunakan permata dari lapisan dalam suatu fungsi. Contoh:

```
require 'json'
require 'order_validator'

def lambda_handler(event:, context:)
  begin
    # Parse the order data from the event body
    order_data = JSON.parse(event['body'] || '{}')

    # Validate and format the order
    validated_order = OrderValidator.validate_order(order_data)

    OrderValidator.format_response(200, {
      message: 'Order validated successfully',
      order: validated_order
    })
  rescue OrderValidator::ValidationError => e
    OrderValidator.format_response(400, {
      error: e.message
    })
  rescue => e
    OrderValidator.format_response(500, {
      error: 'Internal server error'
    })
  end
end
```

Anda dapat menggunakan peristiwa pengujian berikut untuk menjalankan fungsi:

```
{
    "body": "{\"product_id\": \"ABC123\", \"quantity\": 2, \"priority\": \"express
\"}"
}
```

Tanggapan yang diharapkan:

```
{
```

```
  "statusCode": 200,
  "body": "{\"message\":\"Order validated successfully\",\"order\":{\"product_id\":
\"ABC123\",\"quantity\":2,\"shipping_priority\":\"express\"}}"
}
```

Buat layer di Lambda

Anda dapat mempublikasikan layer Anda menggunakan konsol AWS CLI atau Lambda.

AWS CLI

Jalankan AWS CLI perintah publish-layer-version untuk membuat layer Lambda:

```
aws lambda publish-layer-version --layer-name my-layer --zip-file fileb://layer.zip
 --compatible-runtimes ruby4.0
```

Parameter runtime yang kompatibel adalah opsional. Saat ditentukan, Lambda menggunakan parameter ini untuk memfilter lapisan di konsol Lambda.

Console

Untuk membuat lapisan (konsol)

1. Buka halaman Fungsi di konsol Lambda.
2. Pilih Buat lapisan.
3. Pilih Unggah file.zip, lalu unggah arsip.zip yang Anda buat sebelumnya.
4. (Opsional) Untuk runtime yang kompatibel, pilih runtime Ruby yang sesuai dengan versi Ruby yang Anda gunakan untuk membangun layer Anda.
5. Pilih Buat.

Menggunakan permata dari lapisan dalam suatu fungsi

Dalam kode fungsi Anda, Anda harus secara eksplisit meminta setiap permata yang ingin Anda gunakan. Perintah bundler seperti `bundler/setup` dan `Bundler.require` tidak didukung. Berikut cara menggunakan permata dengan benar dari lapisan dalam fungsi Lambda:

```

# Correct: Use explicit requires for each gem
```

```
require 'nokogiri'
require 'httparty'

def lambda_handler(event:, context:)
  # Use the gems directly
  doc = Nokogiri::HTML(event['html'])
  response = HTTParty.get(event['url'])
  # ... rest of your function
end

# Incorrect: These Bundler commands will not work
# require 'bundler/setup'
# Bundler.require
```

Tambahkan layer ke fungsi Anda

AWS CLI

Untuk melampirkan layer ke fungsi Anda, jalankan perintah AWS CLI update-function-configuration. Untuk `--layers` parameter, gunakan lapisan ARN. ARN harus menentukan versi (misalnya,`arn:aws:lambda:us-east-1:123456789012:layer:my-layer:1`). Untuk informasi selengkapnya, lihat Versi lapisan dan lapisan.

```
aws lambda update-function-configuration --function-name my-function --cli-binary-format raw-in-base64-out --layers "arn:aws:lambda:us-east-1:123456789012:layer:my-layer:1"
```

cli-binary-formatOpsi ini diperlukan jika Anda menggunakan AWS CLI versi 2. Untuk menjadikan ini pengaturan default, jalankan`aws configure set cli-binary-format raw-in-base64-out`. Untuk informasi selengkapnya, lihat opsi baris perintah global yang AWS CLI didukung di Panduan AWS Command Line Interface Pengguna untuk Versi 2.

Console

Untuk menambahkan lapisan ke fungsi

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih fungsinya.
3. Gulir ke bawah ke bagian Layers, lalu pilih Add a layer.
4. Di bawah Pilih layer, pilih Custom layers, dan kemudian pilih layer Anda.

> **Note**
>
> Jika Anda tidak menambahkan runtime yang kompatibel saat membuat layer, layer Anda tidak akan tercantum di sini. Anda dapat menentukan lapisan ARN sebagai gantinya.

5. Pilih Tambahkan.

Contoh aplikasi

Untuk contoh lebih lanjut tentang cara menggunakan layer Lambda, lihat contoh aplikasi layer-ruby di repositori Panduan Pengembang. AWS Lambda GitHub Aplikasi ini mencakup lapisan yang berisi pustaka tzinfo. Setelah membuat layer, Anda dapat menerapkan dan memanggil fungsi yang sesuai untuk mengonfirmasi bahwa lapisan berfungsi seperti yang diharapkan.

Menggunakan objek konteks Lambda untuk mengambil informasi Fungsi Acara Ruby

Saat Lambda menjalankan fungsi Anda, ia meneruskan objek konteks ke handler. Objek ini menyediakan metode dan properti yang memberikan informasi tentang lingkungan invokasi, fungsi, dan eksekusi.

Metode konteks

- `get_remaining_time_in_millis` – Mengembalikan jumlah milidetik yang tersisa sebelum waktu eksekusi habis.

Properti konteks

- `function_name` – Nama fungsi Lambda.
- `function_version` – Versi fungsi.
- `invoked_function_arn` – Amazon Resource Name (ARN) yang digunakan untuk memicu fungsi. Menunjukkan jika pemicu menyebutkan nomor versi atau alias.
- `memory_limit_in_mb` – Jumlah memori yang dialokasikan untuk fungsi tersebut.
- `aws_request_id` – Pengidentifikasi permintaan invokasi.
- `log_group_name` – Grup log untuk fungsi.
- `log_stream_name` – Aliran log untuk instans fungsi.
- `deadline_ms`– Tanggal saat eksekusi berakhir, dalam milidetik waktu Unix.
- `identity` – (aplikasi seluler) Informasi tentang identitas Amazon Cognito yang mengesahkan permintaan.
- `client_context`– (aplikasi seluler) Konteks klien yang disediakan untuk Lambda oleh aplikasi klien.

Log dan pantau Fungsi Acara Ruby Lambda

AWS Lambda secara otomatis memonitor Fungsi Peristiwa Lambda atas nama Anda dan mengirim log ke Amazon. CloudWatch Fungsi Anda dilengkapi dengan grup CloudWatch log Log dan aliran log untuk setiap instance fungsi Anda. Lingkungan runtime Lambda mengirimkan detail tentang setiap invokasi ke pengaliran log, dan menyampaikan log serta output lain dari kode fungsi Anda. Untuk informasi selengkapnya, lihat Mengirim log Fungsi Peristiwa Lambda ke Log CloudWatch.

Halaman ini menjelaskan cara menghasilkan keluaran log dari kode fungsi Lambda Anda, dan mengakses log menggunakan AWS Command Line Interface, konsol Lambda, atau konsol. CloudWatch

Bagian-bagian

- Membuat fungsi yang mengembalikan log
- Menggunakan kontrol logging lanjutan Lambda dengan Ruby
- Melihat log di konsol Lambda
- Melihat log di CloudWatch konsol
- Melihat log menggunakan AWS Command Line Interface (AWS CLI)
- Menghapus log
- Bekerja dengan pustaka logger Ruby

Membuat fungsi yang mengembalikan log

Untuk menghasilkan log dari kode fungsi, Anda dapat menggunakan pernyataan `puts`, atau pustaka pencatatan apa pun yang menulis ke `stdout` atau `stderr`. Contoh berikut mencatat nilai variabel lingkungan dan objek peristiwa. Untuk informasi tentang objek konteks yang tersedia untuk handler Anda, lihatthe section called “Konteks”.

Example lambda_function.rb

```
# lambda_function.rb

def handler(event:, context:)
    puts "## ENVIRONMENT VARIABLES"
    puts ENV.to_a
    puts "## EVENT"
    puts event.to_a
```

```
end
```

Example format log

```
START RequestId: 8f507cfc-xmpl-4697-b07a-ac58fc914c95 Version: $LATEST
## ENVIRONMENT VARIABLES
environ({'AWS_LAMBDA_LOG_GROUP_NAME': '/aws/lambda/my-function',
 'AWS_LAMBDA_LOG_STREAM_NAME': '2020/01/31/[$LATEST]3893xmpl7fac4485b47bb75b671a283c',
 'AWS_LAMBDA_FUNCTION_NAME': 'my-function', ...})
## EVENT
{'key': 'value'}
END RequestId: 8f507cfc-xmpl-4697-b07a-ac58fc914c95
REPORT RequestId: 8f507cfc-xmpl-4697-b07a-ac58fc914c95  Duration: 15.74 ms  Billed
 Duration: 147 ms Memory Size: 128 MB Max Memory Used: 56 MB  Init Duration: 130.49 ms
XRAY TraceId: 1-5e34a614-10bdxmplf1fb44f07bc535a1   SegmentId: 07f5xmpl2d1f6f85
 Sampled: true
```

Waktu pengoperasian Ruby melakukan log baris `START`, `END`, dan `REPORT` untuk setiap invokasi. Baris laporan memberikan perincian berikut.

Laporkan bidang data baris

- RequestId – ID permintaan unik untuk invokasi.
- Durasi – Jumlah waktu yang digunakan oleh metode handler fungsi Anda gunakan untuk memproses peristiwa.
- Durasi yang Ditagih – Jumlah waktu yang ditagihkan untuk invokasi.
- Ukuran Memori – Jumlah memori yang dialokasikan untuk fungsi.
- Memori Maks yang Digunakan – Jumlah memori yang digunakan oleh fungsi. Saat pemanggilan berbagi lingkungan eksekusi, Lambda melaporkan memori maksimum yang digunakan di semua pemanggilan. Perilaku ini dapat menghasilkan nilai yang dilaporkan lebih tinggi dari yang diharapkan.
- Durasi Init – Untuk permintaan pertama yang dilayani, lama waktu yang diperlukan runtime untuk memuat fungsi dan menjalankan kode di luar metode handler.
- XRAY TraceId — Untuk permintaan yang dilacak, ID AWS X-Ray jejak.
- SegmentId— Untuk permintaan yang dilacak, ID X-Ray segmen.
- Diambil Sampel – Untuk permintaan yang dilacak, hasil pengambilan sampel.

Untuk log yang lebih rinci, gunakan filethe section called “Bekerja dengan pustaka logger Ruby”.

Menggunakan kontrol logging lanjutan Lambda dengan Ruby

Untuk memberi Anda kontrol lebih besar atas bagaimana log fungsi Anda ditangkap, diproses, dan dikonsumsi, Lambda menawarkan kontrol pencatatan lanjutan dengan Ruby. Untuk Ruby 4.0 dan runtime yang lebih baru, Anda dapat mengonfigurasi opsi logging berikut:

- Format log - pilih antara teks biasa dan format JSON terstruktur untuk log fungsi Anda
- Tingkat log - untuk log dalam format JSON, pilih tingkat detail log yang dikirim Lambda ke CloudWatch Amazon, seperti ERROR, DEBUG, atau INFO
- Grup log - pilih grup CloudWatch log yang dikirimkan oleh fungsi Anda

Untuk informasi selengkapnya tentang opsi pencatatan ini, dan petunjuk tentang cara mengonfigurasi fungsi Anda untuk menggunakannya, lihatthe section called “Mengkonfigurasi kontrol logging lanjutan untuk Lambda Event Functions”.

Untuk mempelajari lebih lanjut tentang menggunakan format log dan opsi tingkat log dengan fungsi Ruby Lambda Anda, lihat panduan di bagian berikut.

Menggunakan log JSON terstruktur dengan Ruby

Jika Anda memilih JSON untuk format log fungsi Anda, Lambda akan mengirim output log oleh pustaka `Logger` standar Ruby CloudWatch ke JSON terstruktur. Setiap objek log JSON berisi setidaknya empat pasangan nilai kunci dengan kunci berikut:

- `"timestamp"`- waktu pesan log dihasilkan
- `"level"`- tingkat log yang ditetapkan untuk pesan
- `"message"`- isi pesan log
- `"requestId"`- ID permintaan unik untuk pemanggilan fungsi

`Logger`Pustaka Ruby juga dapat menambahkan pasangan nilai kunci tambahan seperti `"logger"` ke objek JSON ini.

Contoh di bagian berikut menunjukkan bagaimana output log yang dihasilkan menggunakan `Logger` pustaka Ruby ditangkap di CloudWatch Log saat Anda mengonfigurasi format log fungsi Anda sebagai JSON.

Perhatikan bahwa jika Anda menggunakan `puts` metode untuk menghasilkan output log dasar seperti yang dijelaskan dalamthe section called “Membuat fungsi yang mengembalikan log”, Lambda

akan menangkap output ini sebagai teks biasa, bahkan jika Anda mengonfigurasi format logging fungsi Anda sebagai JSON.

Output log JSON standar menggunakan pustaka Ruby Logger

Contoh cuplikan kode dan keluaran log berikut menunjukkan bagaimana output log standar yang dihasilkan menggunakan `Logger` pustaka Ruby ditangkap di CloudWatch Log saat format log fungsi Anda disetel ke JSON.

Example Contoh kode logging Ruby

```
require 'logger'

def lambda_handler(event:, context:)
  logger = Logger.new($stdout)
  logger.info("Inside the handler function")
end
```

Example Contoh catatan log JSON

```
{
    "timestamp": "2025-10-27T19:17:45.586Z",
    "level": "INFO",
    "message": "Inside the handler function",
    "requestId": "79b4f56e-95b1-4643-9700-2807f4e68189"
}
```

Mencatat parameter tambahan di JSON

Ketika format log fungsi Anda diatur ke JSON, Anda juga dapat mencatat parameter tambahan dengan `Logger` pustaka Ruby dengan meneruskan hash pasangan nilai kunci tambahan ke output log.

Example Contoh kode logging Ruby

```
require 'logger'
require 'json'

def lambda_handler(event:, context:)
  logger = Logger.new($stdout)
  extra_params = { "a" => "b", "b" => [3] }
```

```
  logger.info({ message: "extra parameters example" }.merge(extra_params).to_json)
end
```

Example Contoh catatan log JSON

```
{
    "timestamp": "2025-11-02T15:26:28Z",
    "level": "INFO",
    "message": "extra parameters example",
    "requestId": "3dbd5759-65f6-45f8-8d7d-5bdc79a3bd01",
    "a": "b",
    "b": [
        3
    ]
}
```

Pengecualian logging di JSON

Cuplikan kode berikut menunjukkan bagaimana pengecualian Ruby ditangkap dalam output log fungsi Anda saat Anda mengonfigurasi format log sebagai JSON. Perhatikan bahwa output log yang dihasilkan menggunakan `logger.error` dengan pengecualian ditetapkan ERROR tingkat log.

Example Contoh kode logging Ruby

```
require 'logger'

def lambda_handler(event:, context:)
  logger = Logger.new($stdout)
  begin
    raise "exception"
  rescue => e
    logger.error(e)
  end
end
```

Example Contoh catatan log JSON

```
{
    "timestamp": "2025-11-02T16:18:57Z",
    "level": "ERROR",
    "message": "exception",
    "stackTrace": [
```

```
        "  /var/task/lambda_function.rb:4:in `lambda_handler'"
    ],
    "errorType": "RuntimeError",
    "errorMessage": "exception",
    "requestId": "3f9d155c-0f09-46b7-bdf1-e91dab220855",
    "location": "/var/task/lambda_function.rb:lambda_handler:6"
}
```

Menggunakan penyaringan log-level dengan Ruby

Dengan mengonfigurasi penyaringan tingkat log, Anda dapat memilih untuk mengirim hanya log dari tingkat logging tertentu atau lebih rendah ke Log. CloudWatch Untuk mempelajari cara mengonfigurasi pemfilteran tingkat log untuk fungsi Anda, lihat. the section called “Log-level penyaringan”

AWS Lambda Untuk memfilter log aplikasi Anda sesuai dengan tingkat lognya, fungsi Anda harus menggunakan log berformat JSON. Anda dapat mencapai ini dengan dua cara:

- Buat output log menggunakan `Logger` pustaka Ruby standar dan konfigurasikan fungsi Anda untuk menggunakan pemformatan log JSON. AWS Lambda kemudian memfilter output log Anda menggunakan pasangan nilai `"level"` kunci di objek JSON yang dijelaskan dalam. the section called “Menggunakan log JSON terstruktur dengan Ruby” Untuk mempelajari cara mengonfigurasi format log fungsi Anda, lihatthe section called “Mengkonfigurasi kontrol logging lanjutan untuk Lambda Event Functions”.
- Gunakan pustaka atau metode logging lain untuk membuat log terstruktur JSON dalam kode Anda yang menyertakan pasangan nilai `"level"` kunci yang menentukan tingkat output log.

Anda juga dapat menggunakan `puts` pernyataan untuk menampilkan objek JSON yang berisi pengenal tingkat log. `puts`Pernyataan berikut menghasilkan output JSON diformat di mana tingkat log diatur ke INFO. AWS Lambda akan mengirim objek JSON ke CloudWatch Log jika level logging fungsi Anda diatur ke INFO, DEBUG, atau TRACE.

```
puts '{"msg":"My log message", "level":"info"}'
```

Agar Lambda dapat memfilter log fungsi Anda, Anda juga harus menyertakan pasangan nilai `"timestamp"` kunci dalam keluaran log JSON Anda. Waktu harus ditentukan dalam format stempel waktu RFC 3339 yang valid. Jika Anda tidak menyediakan stempel waktu yang valid, Lambda akan menetapkan log INFO level dan menambahkan stempel waktu untuk Anda.

Menggunakan pustaka logging alternatif

Jika Anda perlu menggunakan versi kustom `logger` pustaka, Anda dapat memasukkannya ke dalam paket penerapan atau dalam lapisan Lambda, dan mengatur `RUBYLIB` variabel lingkungan ke direktori perpustakaan`lib`. Runtime Lambda akan memuat versi Anda alih-alih versi yang dibundel.

Jika kode Anda sudah menggunakan pustaka logging lain untuk menghasilkan log terstruktur JSON, Anda tidak perlu membuat perubahan apa pun. AWS Lambda tidak menyandikan dua kali log apa pun yang sudah dikodekan JSON. Bahkan jika Anda mengonfigurasi fungsi Anda untuk menggunakan format log JSON, output logging Anda muncul CloudWatch dalam struktur JSON yang Anda tentukan.

Melihat log di konsol Lambda

Anda dapat menggunakan konsol Lambda untuk melihat keluaran log setelah Anda menjalankan Fungsi Peristiwa Lambda.

Jika kode Anda dapat diuji dari editor Kode tertanam, Anda menemukan log dalam hasil eksekusi. Saat Anda menggunakan fitur pengujian konsol untuk menjalankan fungsi, Anda menemukan Keluaran Log di bagian Detail.

Melihat log di CloudWatch konsol

Anda dapat menggunakan CloudWatch konsol Amazon untuk melihat log untuk semua pemanggilan Fungsi Peristiwa Lambda.

Untuk melihat log di CloudWatch konsol

1. Buka halaman Grup log di CloudWatch konsol.
2. Pilih grup log untuk fungsi Anda (/aws/lambda/***your-function-name***).
3. Pilih pengaliran log.

Setiap aliran log sesuai dengan instans fungsi Anda. Aliran log muncul saat Anda memperbarui fungsi Lambda, dan saat instance tambahan dibuat untuk menangani pemanggilan bersamaan. Untuk menemukan log untuk pemanggilan tertentu, kami sarankan untuk menginstrumentasi fungsi Anda dengan. AWS X-Ray X-Ray mencatat detail tentang permintaan dan aliran log dalam jejak.

Melihat log menggunakan AWS Command Line Interface (AWS CLI)

AWS CLI Ini adalah alat sumber terbuka yang dapat Anda gunakan untuk berinteraksi dengan AWS layanan menggunakan perintah di shell baris perintah Anda. Untuk menyelesaikan langkah-langkah di bagian ini, Anda harus memiliki AWS CLI versi 2.

Anda dapat menggunakan AWS CLI untuk mengambil log untuk invokasi menggunakan opsi perintah `--log-type`. Respons berisi bidang `LogResult` yang memuat hingga 4 KB log berkode base64 dari invokasi.

Example mengambil ID log

Contoh berikut menunjukkan cara mengambil ID log dari `LogResult` untuk fungsi bernama `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

Anda akan melihat output berikut:

```
{
    "StatusCode": 200,
    "LogResult":
 "U1RBUlQgUmVxdWVzdElkOiA4N2QwNDRiOC1mMTU0LTExZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "ExecutedVersion": "$LATEST"
}
```

Example mendekode log

Pada prompt perintah yang sama, gunakan utilitas `base64` untuk mendekodekan log. Contoh berikut menunjukkan cara mengambil log berkode base64 untuk `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail \
--query 'LogResult' --output text --cli-binary-format raw-in-base64-out | base64 --
decode
```

cli-binary-formatOpsi ini diperlukan jika Anda menggunakan AWS CLI versi 2. Untuk menjadikan ini pengaturan default, jalankan`aws configure set cli-binary-format raw-in-base64-out`. Untuk informasi selengkapnya, lihat opsi baris perintah global yang AWS CLI didukung di Panduan AWS Command Line Interface Pengguna untuk Versi 2.

Anda akan melihat output berikut:

```
START RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Version: $LATEST
"AWS_SESSION_TOKEN": "AgoJb3JpZ2luX2VjELj...", "_X_AMZN_TRACE_ID": "Root=1-5d02e5ca-
f5792818b6fe8368e5b51d50;Parent=191db58857df8395;Sampled=0"",ask/lib:/opt/lib",
END RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8
REPORT RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8  Duration: 79.67 ms      Billed
 Duration: 80 ms         Memory Size: 128 MB     Max Memory Used: 73 MB
```

Utilitas `base64` tersedia di Linux, macOS, dan Ubuntu pada Windows. Pengguna macOS mungkin harus menggunakan `base64 -D`.

Example Skrip get-logs.sh

Pada prompt perintah yang sama, gunakan script berikut untuk mengunduh lima peristiwa log terakhir. Skrip menggunakan `sed` untuk menghapus kutipan dari file output, dan akan tidur selama 15 detik untuk memberikan waktu agar log tersedia. Output mencakup respons dari Lambda dan output dari perintah `get-log-events`.

Salin konten dari contoh kode berikut dan simpan dalam direktori proyek Lambda Anda sebagai `get-logs.sh`.

cli-binary-formatOpsi ini diperlukan jika Anda menggunakan AWS CLI versi 2. Untuk menjadikan ini pengaturan default, jalankan`aws configure set cli-binary-format raw-in-base64-out`. Untuk informasi selengkapnya, lihat opsi baris perintah global yang AWS CLI didukung di Panduan AWS Command Line Interface Pengguna untuk Versi 2.

```
#!/bin/bash
aws lambda invoke --function-name my-function --cli-binary-format raw-in-base64-out --
payload '{"key": "value"}' out
sed -i'' -e 's/"//g' out
sleep 15
aws logs get-log-events --log-group-name /aws/lambda/my-function --log-stream-
name stream1 --limit 5
```

Example macOS dan Linux (khusus)

Dalam prompt perintah yang sama, pengguna macOS dan Linux mungkin perlu menjalankan perintah berikut untuk memastikan skrip dapat dieksekusi.

```
chmod -R 755 get-logs.sh
```

Example mengambil lima log acara terakhir

Pada prompt perintah yang sama, gunakan skrip berikut untuk mendapatkan lima log acara terakhir.

```
./get-logs.sh
```

Anda akan melihat output berikut:

```
{
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}
{
    "events": [
        {
            "timestamp": 1559763003171,
            "message": "START RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf Version:
 $LATEST\n",
            "ingestionTime": 1559763003309
        },
        {
            "timestamp": 1559763003173,
            "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tENVIRONMENT VARIABLES\r{\r  \"AWS_LAMBDA_FUNCTION_VERSION\": \"$LATEST\",
\r ...",
            "ingestionTime": 1559763018353
        },
        {
            "timestamp": 1559763003173,
            "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tEVENT\r{\r  \"key\": \"value\"\r}\n",
            "ingestionTime": 1559763018353
        },
        {
            "timestamp": 1559763003218,
            "message": "END RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf\n",
            "ingestionTime": 1559763018353
        },
        {
            "timestamp": 1559763003218,
            "message": "REPORT RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tDuration: 26.73 ms\tBilled Duration: 27 ms \tMemory Size: 128 MB\tMax Memory Used: 75
 MB\t\n",
```

```
            "ingestionTime": 1559763018353
        }
    ],
    "nextForwardToken": "f/34783877304859518393868359594929986069206639495374241795",
    "nextBackwardToken": "b/34783877303811383369537420289090800615709599058929582080"
}
```

Menghapus log

Grup log tidak terhapus secara otomatis ketika Anda menghapus suatu fungsi. Untuk menghindari penyimpanan log secara tidak terbatas, hapus kelompok log, atau lakukan konfigurasi periode penyimpanan, yang setelahnya log akan dihapus secara otomatis.

Bekerja dengan pustaka logger Ruby

Pustaka logger Ruby mengembalikan log yang disederhanakan yang mudah dibaca. Gunakan utilitas logger untuk menampilkan informasi rinci, pesan, dan kode kesalahan yang terkait dengan fungsi Anda.

```
# lambda_function.rb

require 'logger'

def handler(event:, context:)
  logger = Logger.new($stdout)
  logger.info('## ENVIRONMENT VARIABLES')
  logger.info(ENV.to_a)
  logger.info('## EVENT')
  logger.info(event)
  event.to_a
end
```

Output dari `logger` mencakup tingkat log, stempel waktu, dan ID permintaan.

```
START RequestId: 1c8df7d3-xmpl-46da-9778-518e6eca8125 Version: $LATEST
[INFO]  2020-01-31T22:12:58.534Z    1c8df7d3-xmpl-46da-9778-518e6eca8125    ##
 ENVIRONMENT VARIABLES

[INFO]  2020-01-31T22:12:58.534Z    1c8df7d3-xmpl-46da-9778-518e6eca8125
    environ({'AWS_LAMBDA_LOG_GROUP_NAME': '/aws/lambda/my-function',
 'AWS_LAMBDA_LOG_STREAM_NAME': '2020/01/31/[$LATEST]1bbe51xmplb34a2788dbaa7433b0aa4d',
 'AWS_LAMBDA_FUNCTION_NAME': 'my-function', ...})
```

```
[INFO]  2020-01-31T22:12:58.535Z    1c8df7d3-xmpl-46da-9778-518e6eca8125    ## EVENT

[INFO]  2020-01-31T22:12:58.535Z    1c8df7d3-xmpl-46da-9778-518e6eca8125    {'key':
 'value'}

END RequestId: 1c8df7d3-xmpl-46da-9778-518e6eca8125
REPORT RequestId: 1c8df7d3-xmpl-46da-9778-518e6eca8125  Duration: 2.75 ms   Billed
 Duration: 117 ms Memory Size: 128 MB Max Memory Used: 56 MB  Init Duration: 113.51 ms
XRAY TraceId: 1-5e34a66a-474xmpl7c2534a87870b4370   SegmentId: 073cxmpl3e442861
 Sampled: true
```

Menginstrumentasi kode Ruby di AWS Lambda

Lambda terintegrasi dengan AWS X-Ray untuk memungkinkan Anda melacak, men-debug, dan mengoptimalkan aplikasi Lambda. Anda dapat menggunakan X-Ray untuk melacak permintaan saat melintasi sumber daya dalam aplikasi Anda, dari API frontend hingga penyimpanan dan database di backend. Hanya dengan menambahkan pustaka X-Ray SDK ke konfigurasi build, Anda dapat merekam error dan latensi untuk panggilan apa pun yang dilakukan fungsi ke layanan. AWS

Setelah mengonfigurasi penelusuran aktif, Anda dapat mengamati permintaan tertentu melalui aplikasi Anda. Grafik X-Ray layanan menunjukkan informasi tentang aplikasi Anda dan semua komponennya. Contoh berikut menunjukkan aplikasi dengan dua fungsi. Fungsi utama memproses kejadian dan terkadang mengembalikan kesalahan. Fungsi kedua di bagian atas memproses kesalahan yang muncul di grup log pertama dan menggunakan AWS SDK untuk memanggil X-Ray, Amazon Simple Storage Service (Amazon S3) Service, dan Amazon Logs. CloudWatch



Untuk mengaktifkan penelusuran aktif di Fungsi Peristiwa Lambda Anda dengan konsol, ikuti langkah-langkah berikut:

Untuk mengaktifkan penelusuran aktif

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih Fungsi Acara.
3. Pilih Konfigurasi dan kemudian pilih Alat Pemantauan dan operasi.
4. Di bawah Alat pemantauan tambahan, pilih Edit.

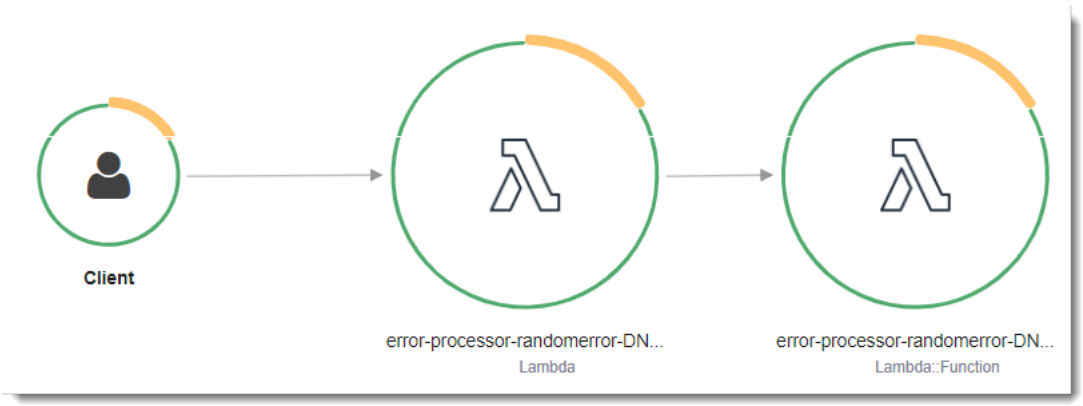
5. Di bawah Sinyal CloudWatch Aplikasi dan AWS X-Ray, pilih Aktifkan untuk jejak layanan Lambda.
6. Pilih Simpan.

> **Harga**
>
> Anda dapat menggunakan X-Ray tracing gratis setiap bulan hingga batas tertentu sebagai bagian dari Tingkat AWS Gratis. Di luar ambang batas itu, X-Ray biaya untuk penyimpanan dan pengambilan jejak. Untuk informasi selengkapnya, lihat harga AWS X-Ray.

Fungsi Acara Anda memerlukan izin untuk mengunggah data jejak ke X-Ray. Saat Anda mengaktifkan penelusuran di konsol Lambda, Lambda menambahkan izin yang diperlukan ke peran eksekusi Fungsi Acara Anda. Atau, tambahkan kebijakan AWSXRayDaemonWriteAccess ke peran eksekusi.

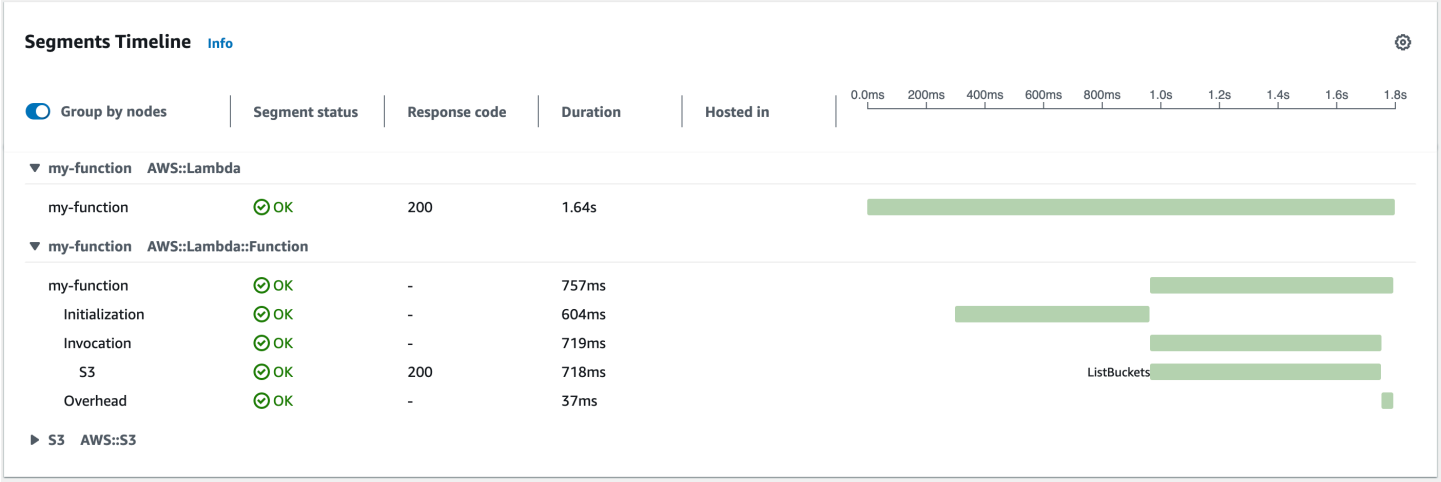
X-Ray tidak melacak semua permintaan ke aplikasi Anda. X-Ray menerapkan algoritma pengambilan sampel untuk memastikan bahwa penelusuran efisien, sambil tetap memberikan sampel yang representatif dari semua permintaan. Tingkat pengambilan sampel adalah 1 permintaan per detik dan 5 persen dari permintaan tambahan. Anda tidak dapat mengonfigurasi laju X-Ray pengambilan sampel untuk fungsi Anda.

Di X-Ray, jejak mencatat informasi tentang permintaan yang diproses oleh satu atau beberapa layanan. Lambda mencatat 2 segmen per jejak, yang menciptakan dua node pada grafik layanan. Gambar berikut menyoroti dua node ini:



Node pertama di sebelah kiri mewakili layanan Lambda, yang menerima permintaan pemanggilan. Node kedua mewakili fungsi Lambda spesifik Anda. Contoh berikut menunjukkan jejak dengan

dua segmen ini. Keduanya bernama fungsi saya, tetapi yang satu memiliki asal `AWS::Lambda` dan yang lainnya memiliki asal usul. `AWS::Lambda::Function` Jika `AWS::Lambda` segmen menunjukkan kesalahan, layanan Lambda mengalami masalah. Jika `AWS::Lambda::Function` segmen menunjukkan kesalahan, fungsi Anda mengalami masalah.



Contoh ini memperluas `AWS::Lambda::Function` segmen untuk menunjukkan tiga subsegmennya.

> **Note**
>
> AWS saat ini menerapkan perubahan pada layanan Lambda. Karena perubahan ini, Anda mungkin melihat perbedaan kecil antara struktur dan konten pesan log sistem dan segmen pelacakan yang dipancarkan oleh Fungsi Peristiwa Lambda yang berbeda di Anda. Akun AWS
> Contoh jejak yang ditunjukkan di sini menggambarkan segmen fungsi gaya lama. Perbedaan antara segmen gaya lama dan baru dijelaskan dalam paragraf berikut.
> Perubahan ini akan diterapkan selama beberapa minggu mendatang, dan semua fungsi di semua Wilayah AWS kecuali China GovCloud dan wilayah akan bertransisi untuk menggunakan pesan log format baru dan segmen pelacakan.

Segmen fungsi gaya lama berisi subsegmen berikut:

- Inisialisasi – Mewakili waktu yang dihabiskan untuk memuat fungsi dan menjalankan kode inisialisasi. Subsegmen ini hanya muncul untuk peristiwa pertama yang diproses oleh setiap instance fungsi Anda.
- Doa - Merupakan waktu yang dihabiskan untuk menjalankan kode handler Anda.

- Overhead - Merupakan waktu yang dihabiskan runtime Lambda untuk mempersiapkan diri untuk menangani acara berikutnya.

Segmen fungsi gaya baru tidak berisi subsegmen. `Invocation` Sebagai gantinya, subsegmen pelanggan dilampirkan langsung ke segmen fungsi. Untuk informasi lebih lanjut tentang struktur segmen fungsi gaya lama dan baru, lihat. the section called “Memahami X-Ray jejak”

Anda dapat melakukan instrumentasi kode penangan untuk merekam metadata dan melacak panggilan downstream. Untuk merekam detail tentang panggilan yang dilakukan handler Anda ke sumber daya dan layanan lain, gunakan X-Ray SDK for Ruby. Untuk mendapatkan SDK, tambahkan paket `aws-xray-sdk` ke dependensi aplikasi Anda.

Example kosong- ruby/function /Gemfile

```
# Gemfile
source 'https://rubygems.org'

gem 'aws-xray-sdk', '0.11.4'
gem 'aws-sdk-lambda', '1.39.0'
gem 'test-unit', '3.3.5'
```

Untuk instrumen klien AWS SDK, memerlukan `aws-xray-sdk/lambda` modul setelah membuat klien dalam kode inisialisasi.

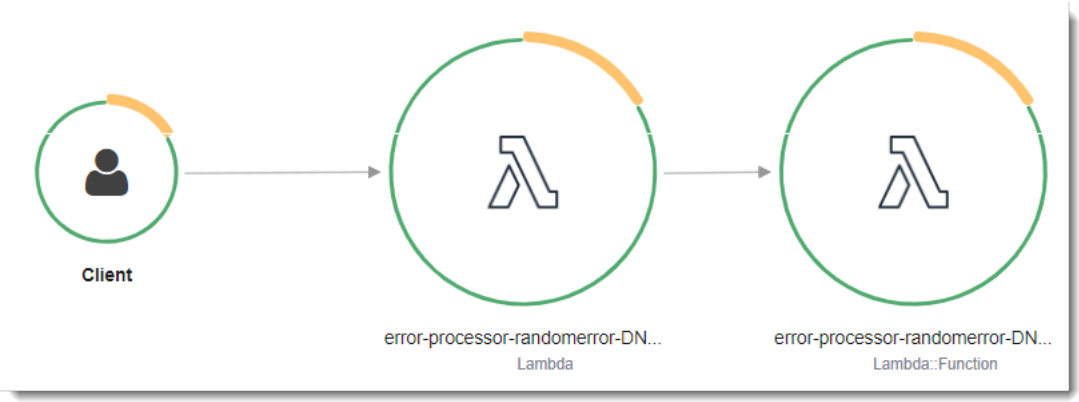
Example kosong- ruby/function /lambda_function.rb — Menelusuri sebuah AWS Klien SDK

```
# lambda_function.rb
require 'logger'
require 'json'
require 'aws-sdk-lambda'
$client = Aws::Lambda::Client.new()
$client.get_account_settings()

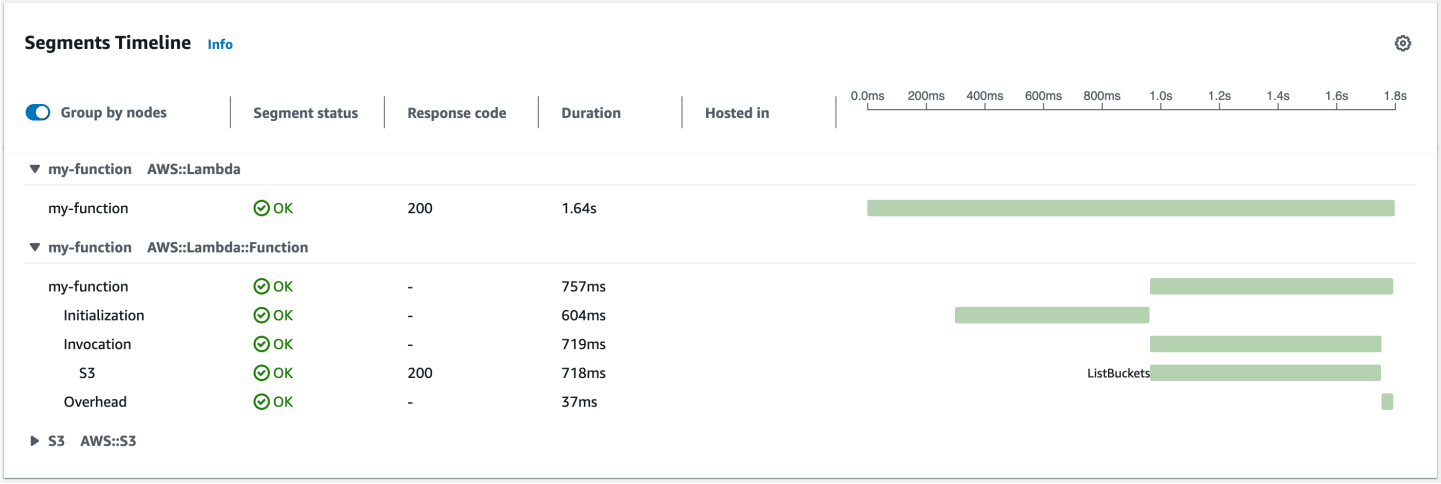
require 'aws-xray-sdk/lambda'

def lambda_handler(event:, context:)
  logger = Logger.new($stdout)
  ...
```

Di X-Ray, jejak mencatat informasi tentang permintaan yang diproses oleh satu atau beberapa layanan. Lambda mencatat 2 segmen per jejak, yang menciptakan dua node pada grafik layanan. Gambar berikut menyoroti dua node ini:



Node pertama di sebelah kiri mewakili layanan Lambda, yang menerima permintaan pemanggilan. Node kedua mewakili fungsi Lambda spesifik Anda. Contoh berikut menunjukkan jejak dengan dua segmen ini. Keduanya bernama fungsi saya, tetapi yang satu memiliki asal `AWS::Lambda` dan yang lainnya memiliki asal usul. `AWS::Lambda::Function` Jika `AWS::Lambda` segmen menunjukkan kesalahan, layanan Lambda mengalami masalah. Jika `AWS::Lambda::Function` segmen menunjukkan kesalahan, fungsi Anda mengalami masalah.



Contoh ini memperluas `AWS::Lambda::Function` segmen untuk menunjukkan tiga subsegmennya.

> **Note**
>
> AWS saat ini menerapkan perubahan pada layanan Lambda. Karena perubahan ini, Anda mungkin melihat perbedaan kecil antara struktur dan konten pesan log sistem dan segmen

pelacakan yang dipancarkan oleh Fungsi Peristiwa Lambda yang berbeda di Anda. Akun AWS
Contoh jejak yang ditunjukkan di sini menggambarkan segmen fungsi gaya lama. Perbedaan antara segmen gaya lama dan baru dijelaskan dalam paragraf berikut.
Perubahan ini akan diterapkan selama beberapa minggu mendatang, dan semua fungsi di semua Wilayah AWS kecuali China GovCloud dan wilayah akan bertransisi untuk menggunakan pesan log format baru dan segmen pelacakan.

Segmen fungsi gaya lama berisi subsegmen berikut:

- Inisialisasi – Mewakili waktu yang dihabiskan untuk memuat fungsi dan menjalankan kode inisialisasi. Subsegmen ini hanya muncul untuk peristiwa pertama yang diproses oleh setiap instance fungsi Anda.
- Doa - Merupakan waktu yang dihabiskan untuk menjalankan kode handler Anda.
- Overhead - Merupakan waktu yang dihabiskan runtime Lambda untuk mempersiapkan diri untuk menangani acara berikutnya.

Segmen fungsi gaya baru tidak berisi subsegmen. `Invocation` Sebagai gantinya, subsegmen pelanggan dilampirkan langsung ke segmen fungsi. Untuk informasi lebih lanjut tentang struktur segmen fungsi gaya lama dan baru, lihat. the section called “Memahami X-Ray jejak”

Anda juga dapat melakukan instrumentasi klien HTTP, merekam kueri SQL, dan membuat subsegmen khusus dengan anotasi dan metadata. Untuk informasi selengkapnya, lihat X-Ray SDK for Ruby AWS X-Ray di Panduan Pengembang.

Bagian-bagian

- Mengaktifkan pelacakan aktif dengan API Lambda
- Mengaktifkan penelusuran aktif dengan CloudFormation
- Menyimpan dependensi runtime dalam satu lapisan

Mengaktifkan pelacakan aktif dengan API Lambda

Untuk mengelola konfigurasi penelusuran dengan AWS CLI atau AWS SDK, gunakan operasi API berikut:

- UpdateFunctionConfiguration

- GetFunctionConfiguration
- CreateFunction

Contoh AWS CLI perintah berikut memungkinkan penelusuran aktif pada fungsi bernama my-function.

```
aws lambda update-function-configuration --function-name my-function \
--tracing-config Mode=Active
```

Mode penelusuran adalah bagian dari konfigurasi khusus versi saat Anda memublikasikan versi fungsi Anda. Anda tidak dapat mengubah mode pelacakan pada versi yang telah diterbitkan.

Mengaktifkan penelusuran aktif dengan CloudFormation

Untuk mengaktifkan penelusuran pada `AWS::Lambda::Function` sumber daya dalam CloudFormation templat, gunakan `TracingConfig` properti.

Example fungsi-inline.yml. — Konfigurasi penelusuran

```
Resources:
  function:
    Type: AWS::Lambda::Function
    Properties:
      TracingConfig:
        Mode: Active
      ...
```

Untuk sumber `AWS::Serverless::Function` daya AWS Serverless Application Model (AWS SAM), gunakan `Tracing` properti.

Example template.yml — Konfigurasi penelusuran

```
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      Tracing: Active
      ...
```

Menyimpan dependensi runtime dalam satu lapisan

Jika Anda menggunakan X-Ray SDK untuk menginstruksikan klien AWS SDK kode fungsi Anda, paket penerapan Anda bisa menjadi cukup besar. Untuk menghindari mengunggah dependensi runtime setiap kali Anda memperbarui kode fungsi, paketkan X-Ray SDK dalam lapisan Lambda.

Contoh berikut menunjukkan `AWS::Serverless::LayerVersion` sumber daya yang menyimpan X-Ray SDK for Ruby.

Example template.yml - Lapisan Dependensi

```
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: function/.
      Tracing: Active
      Layers:
        - !Ref libs
      ...
  libs:
    Type: AWS::Serverless::LayerVersion
    Properties:
      LayerName: blank-ruby-lib
      Description: Dependencies for the blank-ruby sample app.
      ContentUri: lib/.
      CompatibleRuntimes:
        - ruby2.5
```

Dengan konfigurasi ini, Anda memperbarui lapisan pustaka hanya jika Anda mengubah dependensi runtime Anda. Karena paket penerapan fungsi hanya berisi kode Anda, ini dapat membantu mengurangi waktu upload.

Membuat lapisan untuk dependensi memerlukan perubahan build untuk menghasilkan arsip lapisan sebelum penerapan. Untuk contoh pekerjaan, lihat aplikasi sampel blank-ruby.

Membangun Fungsi Acara Lambda dengan Java

Anda dapat menjalankan kode Java di Fungsi AWS Lambda Acara. Lambda menyediakan runtime untuk Java yang menjalankan kode Anda untuk memproses peristiwa. Kode Anda berjalan di lingkungan Amazon Linux yang menyertakan AWS kredensil dari peran AWS Identity and Access Management (IAM) yang Anda kelola.

Lambda mendukung runtime Java berikut ini.

Nama	Pengidentifikasi	Sistem operasi	Tanggal pengusangan	Buat fungsi blok	Pembaruan fungsi blok
Jawa 25	`java25`	Amazon Linux 2023	30 Jun 2029	31 Jul 2029	Agustus 31, 2029
Jawa 21	`java21`	Amazon Linux 2023	30 Jun 2029	31 Jul 2029	Agustus 31, 2029
Jawa 17	`java17.al2023`	Amazon Linux 2023	30 Jun 2029	31 Jul 2029	Agustus 31, 2029
Java 11	`java11.al2023`	Amazon Linux 2023	30 Jun 2029	31 Jul 2029	Agustus 31, 2029
Java 8	`java8.al2023`	Amazon Linux 2023	30 Jun 2029	31 Jul 2029	Agustus 31, 2029
Jawa 17	`java17`	Amazon Linux 2	30 Jun 2027	Jul 29, 2027	Agustus 31, 2027
Java 11	`java11`	Amazon Linux 2	30 Jun 2027	Jul 29, 2027	Agustus 31, 2027
Java 8	`java8.al2`	Amazon Linux 2	30 Jun 2027	Jul 29, 2027	Agustus 31, 2027

AWS menyediakan pustaka berikut untuk fungsi Java. Pustaka ini tersedia melalui repositori pusat Maven.

- com.amazonaws:aws-lambda-java-core (wajib) – Menentukan antarmuka metode handler dan objek konteks yang dilalui oleh runtime ke handler. Jika Anda menentukan jenis input Anda sendiri, ini adalah satu-satunya pustaka yang Anda butuhkan.
- com.amazonaws:aws-lambda-java-events - Jenis masukan untuk acara dari layanan yang memanggil Fungsi Peristiwa Lambda.

- com.amazonaws:aws-lambda-java-log4j2 – Pustaka appender untuk Apache Log4j 2 yang dapat Anda gunakan untuk menambahkan ID permintaan untuk invokasi saat ini ke log fungsi.
- AWS SDK for Java 2.0 — SDK AWS resmi untuk bahasa pemrograman Java.
- AWS Lambda Durable Execution SDK for Java — Bangun aplikasi multi-langkah yang andal dan alur kerja AI yang memeriksa kemajuan, pulih secara otomatis dari kegagalan, dan menjeda eksekusi. Untuk informasi selengkapnya, lihat Fungsi tahan lama Lambda.

Tambahkan library ini ke definisi build Anda sebagai berikut:

Gradle

```
dependencies {
    implementation 'com.amazonaws:aws-lambda-java-core:1.2.2'
    implementation 'com.amazonaws:aws-lambda-java-events:3.11.1'
    runtimeOnly 'com.amazonaws:aws-lambda-java-log4j2:1.5.1'
    implementation 'software.amazon.lambda.durable:aws-durable-execution-sdk-
java:1.0.1'
}
```

Maven

```
  <dependencies>
    <dependency>
      <groupId>com.amazonaws</groupId>
      <artifactId>aws-lambda-java-core</artifactId>
      <version>1.2.2</version>
    </dependency>
    <dependency>
      <groupId>com.amazonaws</groupId>
      <artifactId>aws-lambda-java-events</artifactId>
      <version>3.11.1</version>
    </dependency>
    <dependency>
      <groupId>com.amazonaws</groupId>
      <artifactId>aws-lambda-java-log4j2</artifactId>
      <version>1.5.1</version>
    </dependency>
    <dependency>
      <groupId>software.amazon.lambda.durable</groupId>
      <artifactId>aws-durable-execution-sdk-java</artifactId>
      <version>1.0.1</version>
```

```
    </dependency>
  </dependencies>
```

> **Important**
>
> Jangan gunakan komponen pribadi JDK API, seperti bidang pribadi, metode, atau kelas. Non-public Komponen API dapat berubah atau dihapus dalam pembaruan apa pun, menyebabkan aplikasi Anda rusak.

Untuk informasi tentang serialisasi kustom, lihatthe section called “Serialisasi kustom”.

Untuk membuat Java Event Function

1. Buka Konsol Lambda.
2. Pilih Buat Fungsi Acara.
3. Konfigurasikan pengaturan berikut:
   - Nama fungsi: Masukkan nama untuk fungsi tersebut.
   - Runtime: Pilih Java 25.
4. Pilih Buat Fungsi Acara.

Konsol membuat fungsi Lambda dengan kelas handler bernama `Hello`. Karena Java adalah bahasa kompilasi, Anda tidak dapat melihat atau mengedit kode sumber di konsol Lambda, tetapi Anda dapat memodifikasi konfigurasinya, memanggilnya, dan mengonfigurasi pemicu.

> **Note**
>
> Untuk memulai pengembangan aplikasi di lingkungan lokal Anda, gunakan salah satu contoh aplikasi yang tersedia di GitHub repositori panduan ini.

Kelas `Hello` memiliki fungsi bernama `handleRequest` yang mengambil objek peristiwa dan objek konteks. Ini adalah fungsi handler yang dipanggil Lambda saat fungsi tersebut dipanggil. Runtime fungsi Java mendapatkan peristiwa invokasi dari Lambda dan menyampaikannya ke handler. Dalam konfigurasi fungsi, nilai handler adalah `example.Hello::handleRequest`.

Untuk memperbarui kode fungsi, Anda membuat paket deployment, yang merupakan arsip file .zip yang berisi kode fungsi Anda. Seiring kemajuan pengembangan fungsi, Anda perlu menyimpan kode fungsi Anda dalam kontrol sumber, menambahkan pustaka, dan mengotomatiskan deployment. Mulai dengan membuat paket deployment dan memperbarui kode Anda di baris perintah.

Runtime fungsi melewati objek konteks ke handler, selain peristiwa invokasi. Objek konteks berisi informasi tambahan tentang lingkungan invokasi, fungsi, dan eksekusi. Informasi selengkapnya tersedia dari variabel lingkungan.

Fungsi Lambda Anda dilengkapi dengan grup CloudWatch log Log. Runtime fungsi mengirimkan detail tentang setiap invokasi ke CloudWatch Logs. Detail tersebut menyampaikan log yang dihasilkan fungsi Anda selama invokasi. Jika fungsi mengembalikan kesalahan, Lambda memformat kesalahan dan mengembalikannya ke pemanggil.

Topik

- Tentukan Lambda Event Function handler di Java
- Menyebarkan Fungsi Acara Lambda Java dengan arsip file.zip atau JAR
- Menyebarkan Fungsi Acara Lambda Java dengan gambar kontainer
- Bekerja dengan lapisan untuk Fungsi Acara Java Lambda
- Kustomisasi serialisasi untuk Lambda Java Event Functions
- Kustomisasi perilaku startup runtime Java untuk Fungsi Acara Lambda
- Menggunakan objek konteks Lambda untuk mengambil informasi Java Event Function
- Log dan pantau Fungsi Acara Java Lambda
- Menginstrumentasi kode Java di AWS Lambda
- Contoh aplikasi Java untuk AWS Lambda

Tentukan Lambda Event Function handler di Java

Lambda Event Function handler adalah metode dalam kode fungsi Anda yang memproses peristiwa. Saat fungsi Anda diaktifkan, Lambda menjalankan metode handler. Fungsi Anda berjalan sampai handler mengembalikan respons, keluar, atau waktu habis.

Halaman ini menjelaskan cara bekerja dengan penangan Fungsi Peristiwa Lambda di Java, termasuk opsi untuk penyiapan proyek, konvensi penamaan, dan praktik terbaik. Halaman ini juga menyertakan contoh fungsi Java Lambda yang mengambil informasi tentang pesanan, menghasilkan tanda terima file teks, dan menempatkan file ini di bucket Amazon Simple Storage Service (Amazon S3). Untuk informasi tentang cara menerapkan fungsi Anda setelah menulisnya, lihat the section called “Deploy arsip file .zip” atauthe section called “Deploy gambar kontainer”.

Bagian-bagian

- Menyiapkan proyek handler Java Anda
- Contoh kode Fungsi Acara Java Lambda
- Definisi kelas yang valid untuk penangan Java
- Konvensi penamaan handler
- Mendefinisikan dan mengakses objek peristiwa masukan
- Mengakses dan menggunakan objek konteks Lambda
- Menggunakan AWS SDK for Java v2 di handler Anda
- Mengakses variabel lingkungan
- Menggunakan status global
- Praktik terbaik kode untuk Fungsi Acara Java Lambda

Menyiapkan proyek handler Java Anda

Saat bekerja dengan fungsi Lambda di Java, prosesnya melibatkan penulisan kode Anda, mengompilasinya, dan menyebarkan artefak yang dikompilasi ke Lambda. Anda dapat menginisialisasi proyek Java Lambda dengan berbagai cara. Misalnya, Anda dapat menggunakan alat seperti Maven Archetype untuk fungsi Lambda, perintah AWS SAM CLI sam init, atau bahkan pengaturan proyek Java standar di IDE pilihan Anda, seperti IntelliJ IDEA atau Visual Studio Code. Atau, Anda dapat membuat struktur file yang diperlukan secara manual.

Proyek fungsi Java Lambda yang khas mengikuti struktur umum ini:

```
/project-root
    # src
        # main
            # java
                # example
                    # OrderHandler.java (contains main handler)
                    # <other_supporting_classes>
     # build.gradle OR pom.xml
```

Anda dapat menggunakan Maven atau Gradle untuk membangun proyek dan mengelola dependensi.

Logika handler utama untuk fungsi Anda berada di file Java di bawah direktori. `src/main/java/example` Dalam contoh di halaman ini, kami memberi nama file ini`OrderHandler.java`. Terlepas dari file ini, Anda dapat menyertakan kelas Java tambahan sesuai kebutuhan. Saat menerapkan fungsi Anda ke Lambda, pastikan Anda menentukan kelas Java yang berisi metode pengendali utama yang harus dipanggil Lambda selama pemanggilan.

Contoh kode Fungsi Acara Java Lambda

Contoh berikut kode fungsi Java 21 Lambda mengambil informasi tentang pesanan, menghasilkan tanda terima file teks, dan menempatkan file ini di bucket Amazon S3.

Example **`OrderHandler.java`**Fungsi Acara Lambda

```
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import software.amazon.awssdk.core.sync.RequestBody;
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.model.PutObjectRequest;
import software.amazon.awssdk.services.s3.model.S3Exception;

import java.nio.charset.StandardCharsets;

/**
 * Lambda handler for processing orders and storing receipts in S3.
 */
public class OrderHandler implements RequestHandler<OrderHandler.Order, String> {

    private static final S3Client S3_CLIENT = S3Client.builder().build();
```

```
    /**
     * Record to model the input event.
     */
    public record Order(String orderId, double amount, String item) {}

    @Override
    public String handleRequest(Order event, Context context) {
        try {
            // Access environment variables
            String bucketName = System.getenv("RECEIPT_BUCKET");
            if (bucketName == null || bucketName.isEmpty()) {
                throw new IllegalArgumentException("RECEIPT_BUCKET environment variable
is not set");
            }

            // Create the receipt content and key destination
            String receiptContent = String.format("OrderID: %s\nAmount: $%.2f\nItem:
%s",
                    event.orderId(), event.amount(), event.item());
            String key = "receipts/" + event.orderId() + ".txt";

            // Upload the receipt to S3
            uploadReceiptToS3(bucketName, key, receiptContent);

            context.getLogger().log("Successfully processed order " + event.orderId() +
                    " and stored receipt in S3 bucket " + bucketName);
            return "Success";

        } catch (Exception e) {
            context.getLogger().log("Failed to process order: " + e.getMessage());
            throw new RuntimeException(e);
        }
    }

    private void uploadReceiptToS3(String bucketName, String key, String
receiptContent) {
        try {
            PutObjectRequest putObjectRequest = PutObjectRequest.builder()
                    .bucket(bucketName)
                    .key(key)
                    .build();

            // Convert the receipt content to bytes and upload to S3
```

```
            S3_CLIENT.putObject(putObjectRequest,
 RequestBody.fromBytes(receiptContent.getBytes(StandardCharsets.UTF_8)));
        } catch (S3Exception e) {
            throw new RuntimeException("Failed to upload receipt to S3: " +
 e.awsErrorDetails().errorMessage(), e);
        }
    }
}
```

`OrderHandler.java`File ini berisi bagian kode berikut:

- `package example`: Di Java, ini bisa apa saja, tetapi harus sesuai dengan struktur direktori proyek Anda. Di sini, kita gunakan `package example` karena struktur direktori adalah`src/main/java/example`.
- `import`pernyataan: Gunakan ini untuk mengimpor kelas Java yang diperlukan oleh fungsi Lambda Anda.
- `public class OrderHandler ...`: Ini mendefinisikan kelas Java Anda, dan harus menjadi definisi kelas yang valid.
- `private static final S3Client S3_CLIENT ...`: Ini menginisialisasi klien S3 di luar salah satu metode kelas. Hal ini menyebabkan Lambda menjalankan kode ini selama fase inisialisasi.
- `public record Order ...`: Tentukan bentuk peristiwa input yang diharapkan dalam catatan Java kustom ini.
- `public String handleRequest(Order event, Context context)`: Ini adalah metode handler utama, yang berisi logika aplikasi utama Anda.
- `private void uploadReceiptToS3(...) {}`: Ini adalah metode helper yang direferensikan oleh metode `handleRequest` handler utama.

Contoh build.gradle dan file pom.xml

Berikut `build.gradle` atau `pom.xml` file menyertai fungsi ini.

build.gradle

```
plugins {
    id 'java'
}
```

```
repositories {
    mavenCentral()
}

dependencies {
    implementation 'com.amazonaws:aws-lambda-java-core:1.2.3'
    implementation 'software.amazon.awssdk:s3:2.28.29'
    implementation 'org.slf4j:slf4j-nop:2.0.16'
}

task buildZip(type: Zip) {
    from compileJava
    from processResources
    into('lib') {
        from configurations.runtimeClasspath
    }
}

java {
    sourceCompatibility = JavaVersion.VERSION_21
    targetCompatibility = JavaVersion.VERSION_21
}

build.dependsOn buildZip
```

pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/
maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.example</groupId>
    <artifactId>example-java</artifactId>
    <packaging>jar</packaging>
    <version>1.0-SNAPSHOT</version>
    <name>example-java-function</name>
    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.source>21</maven.compiler.source>
        <maven.compiler.target>21</maven.compiler.target>
    </properties>
    <dependencies>
```

```
        <dependency>
            <groupId>com.amazonaws</groupId>
            <artifactId>aws-lambda-java-core</artifactId>
            <version>1.2.3</version>
        </dependency>
        <dependency>
            <groupId>software.amazon.awssdk</groupId>
            <artifactId>s3</artifactId>
            <version>2.28.29</version>
        </dependency>
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-nop</artifactId>
            <version>2.0.16</version>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <artifactId>maven-surefire-plugin</artifactId>
                <version>3.5.2</version>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-shade-plugin</artifactId>
                <version>3.4.1</version>
                <configuration>
                    <createDependencyReducedPom>false</createDependencyReducedPom>
                    <filters>
                        <filter>
                            <artifact>*:*</artifact>
                            <excludes>
                                <exclude>META-INF/*</exclude>
                                <exclude>META-INF/versions/**</exclude>
                            </excludes>
                        </filter>
                    </filters>
                </configuration>
                <executions>
                    <execution>
                        <phase>package</phase>
                        <goals>
                            <goal>shade</goal>
```

```
                        </goals>
                    </execution>
                </executions>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.13.0</version>
                <configuration>
                    <release>21</release>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

Agar fungsi ini berfungsi dengan baik, peran pelaksanaannya harus memungkinkan `s3:PutObject` tindakan. Juga, pastikan bahwa Anda mendefinisikan variabel `RECEIPT_BUCKET` lingkungan. Setelah pemanggilan berhasil, bucket Amazon S3 harus berisi file tanda terima.

> **Note**
>
> Fungsi ini mungkin memerlukan pengaturan konfigurasi tambahan agar berhasil berjalan tanpa waktu habis. Kami merekomendasikan untuk mengonfigurasi memori 256 MB, dan batas waktu 10 detik. Doa pertama mungkin membutuhkan waktu ekstra karena awal yang dingin. Pemanggilan selanjutnya harus berjalan lebih cepat karena penggunaan kembali lingkungan eksekusi.

Definisi kelas yang valid untuk penangan Java

Untuk menentukan kelas Anda, pustaka aws-lambda-java-core mendefinisikan dua antarmuka untuk metode handler. Gunakan antarmuka yang disediakan untuk menyederhanakan konfigurasi handler dan memvalidasi tanda tangan metode pada waktu kompilasi.

- com.amazonaws.services.lambda.runtime. RequestHandler
- com.amazonaws.services.lambda.runtime. RequestStreamHandler

Antarmuka `RequestHandler` adalah jenis generik yang mengambil dua parameter: jenis input dan jenis output. Kedua jenis tersebut harus berupa objek. Dalam contoh ini, `OrderHandler` kelas kami mengimplementasikan`RequestHandler<OrderHandler.Order, String>`. Tipe input adalah `Order` catatan yang kita definisikan di dalam kelas, dan tipe outputnya adalah`String`.

```
public class OrderHandler implements RequestHandler<OrderHandler.Order, String> {
    ...
}
```

Saat Anda menggunakan antarmuka ini, runtime Java mendeserialisasi acara ke dalam objek dengan tipe input, dan membuat serial output ke dalam teks. Gunakan antarmuka ini saat serialisasi bawaan bekerja dengan tipe input dan output Anda.

Untuk menggunakan serialisasi Anda sendiri, Anda dapat mengimplementasikan `RequestStreamHandler` antarmuka. Dengan antarmuka ini, Lambda melewatkan aliran input dan aliran output ke handler Anda. Handler membaca byte dari aliran input, menuliskan aliran output, dan mengembalikan pembatalan. Untuk contoh ini menggunakan runtime Java 21, lihat HandlerStream.java.

Jika Anda bekerja hanya dengan tipe dasar dan generik (yaitu`String`,, `IntegerList`, atau`Map`) dalam fungsi Java Anda, Anda tidak perlu mengimplementasikan antarmuka. Misalnya, jika fungsi Anda mengambil `Map<String, String>` input dan mengembalikan a`String`, definisi kelas dan tanda tangan handler Anda mungkin terlihat seperti berikut:

```
public class ExampleHandler {
    public String handleRequest(Map<String, String> input, Context context) {
        ...
    }
}
```

Selain itu, ketika Anda tidak mengimplementasikan antarmuka, objek konteks adalah opsional. Misalnya, definisi kelas dan tanda tangan handler Anda mungkin terlihat seperti berikut:

```
public class NoContextHandler {
   public String handleRequest(Map<String, String> input) {
        ...
   }
}
```

Konvensi penamaan handler

Untuk fungsi Lambda di Java, jika Anda menerapkan salah satu `RequestHandler` atau `RequestStreamHandler` antarmuka, metode handler utama Anda harus diberi nama. `handleRequest` Juga, sertakan `@Override` tag di atas `handleRequest` metode Anda. Saat Anda menerapkan fungsi ke Lambda, tentukan pengendali utama dalam konfigurasi fungsi Anda dalam format berikut:

- *<package>*. *<Class>*- Sebagai contoh,`example.OrderHandler`.

Untuk fungsi Lambda di Java yang tidak mengimplementasikan `RequestStreamHandler` antarmuka `RequestHandler` atau, Anda dapat menggunakan nama apa pun untuk handler. Saat Anda menerapkan fungsi ke Lambda, tentukan pengendali utama dalam konfigurasi fungsi Anda dalam format berikut:

- *<package>*. *<Class>*:: *<handler_method_name>* — Misalnya,`example.Handler::mainHandler`.

Mendefinisikan dan mengakses objek peristiwa masukan

JSON adalah format input yang paling umum dan standar untuk fungsi Lambda. Dalam contoh ini, fungsi mengharapkan input yang mirip dengan yang berikut:

```
{
    "orderId": "12345",
    "amount": 199.99,
    "item": "Wireless Headphones"
}
```

Saat bekerja dengan fungsi Lambda di Java 17 atau yang lebih baru, Anda dapat menentukan bentuk peristiwa input yang diharapkan sebagai catatan Java. Dalam contoh ini, kita mendefinisikan catatan dalam `OrderHandler` kelas untuk mewakili `Order` objek:

```
public record Order(String orderId, double amount, String item) {}
```

Catatan ini cocok dengan bentuk input yang diharapkan. Setelah Anda menentukan catatan Anda, Anda dapat menulis tanda tangan handler yang mengambil input JSON yang sesuai dengan definisi rekaman. Runtime Java secara otomatis deserialisasi JSON ini menjadi objek Java. Anda kemudian

dapat mengakses bidang objek. Misalnya, `event.orderId` mengambil nilai dari `orderId` dari input asli.

> **Note**
>
> Catatan Java adalah fitur runtime Java 17 dan yang lebih baru saja. Di semua runtime Java, Anda dapat menggunakan kelas untuk mewakili data peristiwa. Dalam kasus seperti itu, Anda dapat menggunakan perpustakaan seperti jackson untuk deserialisasi input JSON.

Jenis acara masukan lainnya

Ada banyak kemungkinan peristiwa input untuk fungsi Lambda di Java:

- `Integer`,`Long`,`Double`, dan jenis numerik lainnya — Acara ini adalah angka tanpa format tambahan — misalnya,. `3.5` Runtime Java mengubah nilai menjadi objek dari tipe yang ditentukan.
- `String` – Peristiwa tersebut merupakan string JSON, termasuk kutipan—misalnya,`"My string"`. Runtime mengubah nilai menjadi `String` objek tanpa tanda kutip.
- `List<Integer>`,`List<String>`,`List<Object>`, dan jenis serupa - Acara ini adalah array JSON. Runtime mendeserialisasikannya menjadi objek dengan jenis atau antarmuka yang ditentukan.
- `InputStream` – Peristiwa adalah segala jenis JSON. Runtime menyampaikan aliran byte dokumen ke handler tanpa modifikasi. Anda mendeserialisasi input dan menuliskan output ke aliran output.
- Jenis pustaka — Untuk acara yang dikirim oleh AWS layanan lain, gunakan tipe di pustaka aws-lambda-java-events. Misalnya, jika fungsi Lambda Anda dipanggil oleh Amazon Simple Queue Service (SQS), gunakan objek sebagai input. `SQSEvent`

Mengakses dan menggunakan objek konteks Lambda

Objek konteks Lambda berisi informasi tentang lingkungan pemanggilan, fungsi, dan eksekusi. Dalam contoh ini, objek konteks bertipe`com.amazonaws.services.lambda.runtime.Context`, dan merupakan argumen kedua dari fungsi pengendali utama.

```
public String handleRequest(Order event, Context context) {
    ...
}
```

Jika kelas Anda mengimplementasikan RequestStreamHandlerantarmuka RequestHandleratau, objek konteks adalah argumen yang diperlukan. Jika tidak, objek konteks adalah opsional. Untuk informasi selengkapnya tentang tanda tangan handler yang diterima valid, lihat. the section called "Definisi kelas yang valid untuk penangan Java"

Jika Anda melakukan panggilan ke layanan lain menggunakan AWS SDK, objek konteks diperlukan di beberapa area utama. Misalnya, untuk menghasilkan log fungsi untuk Amazon CloudWatch, Anda dapat menggunakan `context.getLogger()` metode untuk mendapatkan `LambdaLogger` objek untuk logging. Dalam contoh ini, kita dapat menggunakan logger untuk mencatat pesan kesalahan jika pemrosesan gagal karena alasan apa pun:

```
context.getLogger().log("Failed to process order: " + e.getMessage());
```

Di luar logging, Anda juga dapat menggunakan objek konteks untuk pemantauan fungsi. Untuk informasi lebih lanjut tentang objek konteks, lihatthe section called "Konteks".

Menggunakan AWS SDK for Java v2 di handler Anda

Seringkali, Anda akan menggunakan fungsi Lambda untuk berinteraksi dengan atau membuat pembaruan ke sumber daya lain AWS . Cara termudah untuk berinteraksi dengan sumber daya ini adalah dengan menggunakan AWS SDK for Java v2.

> **Note**
>
> AWS SDK for Java (v1) berada dalam mode pemeliharaan, dan akan mencapai akhir dukungan pada 31 Desember 2025. Kami menyarankan Anda hanya menggunakan AWS SDK for Java v2 ke depan.

Untuk menambahkan dependensi SDK ke fungsi Anda, tambahkan dependensi SDK di Gradle atau `build.gradle` `pom.xml` file untuk Maven. Kami sarankan hanya menambahkan pustaka yang Anda butuhkan untuk fungsi Anda. Dalam contoh kode sebelumnya, kita menggunakan `software.amazon.awssdk.services.s3` perpustakaan. Di Gradle, Anda dapat menambahkan dependensi ini dengan menambahkan baris berikut di bagian dependensi: `build.gradle`

```
implementation 'software.amazon.awssdk:s3:2.28.29'
```

Di Maven, tambahkan baris berikut di `<dependencies>` bagian Anda: `pom.xml`

```
    <dependency>
        <groupId>software.amazon.awssdk</groupId>
        <artifactId>s3</artifactId>
        <version>2.28.29</version>
    </dependency>
```

> **Note**
>
> Ini mungkin bukan versi SDK terbaru. Pilih versi SDK yang sesuai untuk aplikasi Anda.

Kemudian, impor dependensi langsung di kelas Java Anda:

```
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.model.PutObjectRequest;
import software.amazon.awssdk.services.s3.model.S3Exception;
```

Kode contoh kemudian menginisialisasi klien Amazon S3 sebagai berikut:

```
private static final S3Client S3_CLIENT = S3Client.builder().build();
```

Dalam contoh ini, kami menginisialisasi klien Amazon S3 kami di luar fungsi pengendali utama untuk menghindari keharusan menginisialisasi setiap kali kami menjalankan fungsi kami. Setelah menginisialisasi klien SDK, Anda dapat menggunakannya untuk berinteraksi dengan layanan lain AWS . Kode contoh memanggil Amazon S3 `PutObject` API sebagai berikut:

```
PutObjectRequest putObjectRequest = PutObjectRequest.builder()
    .bucket(bucketName)
    .key(key)
    .build();

// Convert the receipt content to bytes and upload to S3
S3_CLIENT.putObject(putObjectRequest,
 RequestBody.fromBytes(receiptContent.getBytes(StandardCharsets.UTF_8)));
```

Mengakses variabel lingkungan

Dalam kode handler Anda, Anda dapat mereferensikan variabel lingkungan apa pun dengan menggunakan `System.getenv()` metode ini. Dalam contoh ini, kami mereferensikan variabel `RECEIPT_BUCKET` lingkungan yang ditentukan menggunakan baris kode berikut:

```
String bucketName = System.getenv("RECEIPT_BUCKET");
if (bucketName == null || bucketName.isEmpty()) {
    throw new IllegalArgumentException("RECEIPT_BUCKET environment variable is not
 set");
}
```

Menggunakan status global

Lambda menjalankan kode statis Anda dan konstruktor kelas selama fase inisialisasi sebelum menjalankan fungsi Anda untuk pertama kalinya. Sumber daya yang dibuat selama inisialisasi tetap berada di memori di antara pemanggilan, sehingga Anda dapat menghindari keharusan membuatnya setiap kali Anda menjalankan fungsi Anda.

Dalam kode contoh, kode inisialisasi klien S3 berada di luar metode handler utama. Runtime menginisialisasi klien sebelum fungsi menangani peristiwa pertamanya, dan klien tetap tersedia untuk digunakan kembali di semua pemanggilan.

Praktik terbaik kode untuk Fungsi Acara Java Lambda

Patuhi pedoman dalam daftar berikut untuk menggunakan praktik pengkodean terbaik saat membangun fungsi Lambda Anda:

- Pisahkan handler Lambda dari logika inti Anda. Dengan pendekatan ini, Anda dapat membuat fungsi yang lebih dapat diuji unit.
- Kontrol dependensi dalam paket penerapan fungsi Anda. Lingkungan AWS Lambda eksekusi berisi sejumlah pustaka. Untuk mengaktifkan serangkaian fitur dan pembaruan keamanan terbaru, Lambda memperbarui pustaka ini secara berkala. Pembaruan ini mungkin memperkenalkan perubahan halus pada perilaku fungsi Lambda Anda. Untuk memiliki kendali penuh atas dependensi yang digunakan fungsi Anda, kemas semua dependensi Anda dengan paket deployment Anda.
- Minimalkan kompleksitas dependensi Anda. Utamakan memilih kerangka kerja lebih sederhana yang cepat dimuat dalam memulai lingkungan eksekusi. Misalnya, utamakan kerangka kerja injeksi dependensi Java (IoC) yang lebih sederhana seperti Dagger atau Guice, daripada yang lebih kompleks seperti Spring Framework.
- Minimalkan ukuran paket penerapan Anda sesuai kebutuhan runtime-nya. Ini mengurangi jumlah waktu yang diperlukan untuk paket penerapan Anda untuk diunduh dan dibongkar sebelum pemanggilan. Untuk fungsi yang ditulis di Java, hindari mengunggah seluruh pustaka AWS SDK sebagai bagian dari paket penerapan Anda. Sebaliknya, secara selektif bergantung pada modul

yang mengambil komponen SDK yang Anda butuhkan (misalnya, DynamoDB, modul SDK Amazon S3 dan pustaka inti Lambda di situs web). GitHub

Manfaatkan penggunaan kembali lingkungan eksekusi untuk meningkatkan kinerja fungsi Anda. Inisialisasi klien SDK dan koneksi basis data di luar fungsi handler, dan lakukan caching aset statis secara lokal di direktori `/tmp`. Invokasi selanjutnya yang diproses oleh instans yang sama dari fungsi Anda dapat menggunakan kembali sumber daya ini. Ini menghemat biaya dengan mengurangi waktu pengoperasian fungsi.

Untuk menghindari potensi kebocoran data di seluruh invokasi, jangan menggunakan lingkungan eksekusi untuk menyimpan data pengguna, peristiwa, atau informasi lainnya implikasi keamanan. Jika fungsi Anda bergantung pada status yang dapat disenyapkan yang tidak dapat disimpan dalam memori di dalam handler, pertimbangkan untuk membuat fungsi terpisah atau versi terpisah dari fungsi untuk setiap pengguna.

Gunakan arahan keep-alive untuk mempertahankan koneksi yang persisten. Lambda membersihkan koneksi idle dari waktu ke waktu. Mencoba menggunakan ulang koneksi idle saat mengidentifikasi suatu fungsi akan menyebabkan kesalahan koneksi. Untuk mempertahankan koneksi yang persisten, gunakan arahan tetap aktif yang berkaitan dengan runtime Anda. Sebagai contoh, lihat Reusing Connections with Keep-Alive in Node.js.

Gunakan variabel lingkungan untuk meneruskan parameter operasional ke fungsi Anda. Misalnya, jika Anda ingin menulis ke bucket Amazon S3 alih-alih melakukan hard-coding nama bucket yang Anda tulis, konfigurasikan nama bucket sebagai variabel lingkungan.

Hindari menggunakan pemanggilan rekursif dalam fungsi Lambda Anda, di mana fungsi memanggil dirinya sendiri atau memulai proses yang dapat memanggil fungsi lagi. Hal ini dapat menyebabkan volume invokasi fungsi yang tidak diinginkan dan peningkatan biaya. Jika Anda melihat volume pemanggilan yang tidak diinginkan, setel konkurensi fungsi cadangan untuk `0` segera membatasi semua pemanggilan ke fungsi, saat Anda memperbarui kode.

Jangan gunakan API non-publik yang tidak terdokumentasi dalam kode fungsi Lambda Anda. Untuk runtime AWS Lambda terkelola, Lambda secara berkala menerapkan pembaruan keamanan dan fungsional ke API internal Lambda. Pembaruan API internal ini mungkin tidak kompatibel ke belakang, yang menyebabkan konsekuensi yang tidak diinginkan seperti kegagalan pemanggilan jika fungsi Anda memiliki ketergantungan pada API non-publik ini. Lihat referensi API untuk daftar API yang tersedia untuk umum.

Tulis kode idempoten. Menulis kode idempoten untuk fungsi Anda memastikan bahwa peristiwa duplikat ditangani dengan cara yang sama. Kode Anda harus memvalidasi peristiwa dengan benar dan menangani peristiwa duplikat dengan anggun. Untuk informasi selengkapnya, lihat Bagaimana cara membuat fungsi Lambda saya idempoten? .

- Hindari menggunakan cache DNS Java. Fungsi Lambda sudah menyimpan respons DNS cache. Jika Anda menggunakan cache DNS lain, Anda mungkin mengalami batas waktu koneksi.

  `java.util.logging.Logger`Kelas secara tidak langsung dapat mengaktifkan cache DNS JVM. Untuk mengganti pengaturan default, setel networkaddress.cache.ttl ke 0 sebelum menginisialisasi. `logger` Contoh:

  ```
  public class MyHandler {
    // first set TTL property
    static{
     java.security.Security.setProperty("networkaddress.cache.ttl" , "0");
    }
   // then instantiate logger
    var logger = org.apache.logging.log4j.LogManager.getLogger(MyHandler.class);
  }
  ```

- Kurangi waktu yang dibutuhkan Lambda untuk membuka paket deployment yang ditulis di Java dengan meletakkan file `.jar` dependensi Anda dalam direktori/lib terpisah. Ini lebih cepat daripada menempatkan semua kode fungsi Anda dalam satu toples dengan sejumlah besar `.class` file. Lihat Menyebarkan Fungsi Acara Lambda Java dengan arsip file.zip atau JAR untuk instruksi.

Menyebarkan Fungsi Acara Lambda Java dengan arsip file.zip atau JAR

Kode AWS Lambda fungsi Anda terdiri dari skrip atau program yang dikompilasi dan dependensinya. Gunakan paket deployment untuk men-deploy fungsi kode Anda ke Lambda. Lambda mendukung dua tipe paket deployment: gambar kontainer dan arsip file .zip.

Halaman ini menjelaskan cara membuat paket deployment Anda sebagai file.zip atau file Jar, dan kemudian menggunakan paket deployment untuk menyebarkan kode fungsi Anda untuk AWS Lambda menggunakan (). AWS Command Line Interface AWS CLI

> **Important**
>
> Java 25 memperkenalkan dukungan untuk Ahead-of-Time cache (AOT). Kami sangat menyarankan untuk tidak menggunakan cache AOT saat menerapkan fungsi Anda sebagai arsip file.zip atau JAR, karena cache dapat menyebabkan perilaku tak terduga saat Lambda memperbarui runtime yang dikelola. Untuk informasi lebih lanjut, lihatAhead-of-Time (AOT) dan cache CDS.

Bagian-bagian

- Prasyarat
- Alat dan pustaka
- Membangun paket deployment dengan Gradle
- Menggunakan lapisan untuk dependensi
- Membangun paket deployment dengan Maven
- Mengunggah paket penerapan dengan konsol Lambda
- Mengunggah paket penerapan dengan AWS CLI
- Mengunggah paket penerapan dengan AWS SAM

Prasyarat

AWS CLI Ini adalah alat sumber terbuka yang dapat Anda gunakan untuk berinteraksi dengan AWS layanan menggunakan perintah di shell baris perintah Anda. Untuk menyelesaikan langkah-langkah di bagian ini, Anda harus memiliki AWS CLI versi 2.

Alat dan pustaka

AWS menyediakan pustaka berikut untuk fungsi Java. Pustaka ini tersedia melalui repositori pusat Maven.

- com.amazonaws:aws-lambda-java-core (wajib) – Menentukan antarmuka metode handler dan objek konteks yang dilalui oleh runtime ke handler. Jika Anda menentukan jenis input Anda sendiri, ini adalah satu-satunya pustaka yang Anda butuhkan.
- com.amazonaws:aws-lambda-java-events - Jenis masukan untuk acara dari layanan yang memanggil Fungsi Peristiwa Lambda.
- com.amazonaws:aws-lambda-java-log4j2 – Pustaka appender untuk Apache Log4j 2 yang dapat Anda gunakan untuk menambahkan ID permintaan untuk invokasi saat ini ke log fungsi.
- AWS SDK for Java 2.0 — SDK AWS resmi untuk bahasa pemrograman Java.
- AWS Lambda Durable Execution SDK for Java — Bangun aplikasi multi-langkah yang andal dan alur kerja AI yang memeriksa kemajuan, pulih secara otomatis dari kegagalan, dan menjeda eksekusi. Untuk informasi selengkapnya, lihat Fungsi tahan lama Lambda.

Tambahkan library ini ke definisi build Anda sebagai berikut:

Gradle

```
dependencies {
    implementation 'com.amazonaws:aws-lambda-java-core:1.2.2'
    implementation 'com.amazonaws:aws-lambda-java-events:3.11.1'
    runtimeOnly 'com.amazonaws:aws-lambda-java-log4j2:1.5.1'
    implementation 'software.amazon.lambda.durable:aws-durable-execution-sdk-
java:1.0.1'
}
```

Maven

```
  <dependencies>
    <dependency>
      <groupId>com.amazonaws</groupId>
      <artifactId>aws-lambda-java-core</artifactId>
      <version>1.2.2</version>
    </dependency>
    <dependency>
      <groupId>com.amazonaws</groupId>
```

```
      <artifactId>aws-lambda-java-events</artifactId>
      <version>3.11.1</version>
    </dependency>
    <dependency>
      <groupId>com.amazonaws</groupId>
      <artifactId>aws-lambda-java-log4j2</artifactId>
      <version>1.5.1</version>
    </dependency>
    <dependency>
      <groupId>software.amazon.lambda.durable</groupId>
      <artifactId>aws-durable-execution-sdk-java</artifactId>
      <version>1.0.1</version>
    </dependency>
  </dependencies>
```

Untuk membuat paket deployment, kompilasikan kode fungsi dan dependensi Anda menjadi file .zip atau Java Archive (JAR). Untuk Gradle, gunakan tipe build `Zip`. Untuk Apache Maven, gunakan plugin Maven Shade. Untuk mengunggah paket penerapan Anda, gunakan konsol Lambda, API Lambda, atau (). AWS Serverless Application Model AWS SAM

> **Note**
>
> Agar ukuran paket deployment Anda tetap kecil, kemas dependensi fungsi Anda dalam lapisan. Lapisan memungkinkan Anda mengelola dependensi secara mandiri, dapat digunakan oleh beberapa fungsi, dan dapat dibagikan ke akun lain. Untuk informasi selengkapnya, lihat the section called “Lapisan”.

Membangun paket deployment dengan Gradle

Untuk membuat paket penerapan dengan kode dan dependensi fungsi Anda di Gradle, gunakan tipe build. `Zip` Berikut adalah contoh dari contoh file build.gradle lengkap:

Example build.gradle – Membangun tugas

```
task buildZip(type: Zip) {
    into('lib') {
        from(jar)
        from(configurations.runtimeClasspath)
    }
```

```
}
```

Konfigurasi bangunan ini menghasilkan paket deployment di direktori `build/distributions`. Dalam `into('lib')` pernyataan, `jar` tugas merakit arsip jar yang berisi kelas utama Anda ke dalam folder bernama`lib`. Selain itu, `configurations.runtimeClassPath` tugas menyalin pustaka dependensi dari classpath build ke folder yang sama. `lib`

Example build.gradle – Dependensi

```
dependencies {
    ...
    implementation 'com.amazonaws:aws-lambda-java-core:1.2.2'
    implementation 'com.amazonaws:aws-lambda-java-events:3.11.1'
    implementation 'org.apache.logging.log4j:log4j-api:2.17.1'
    implementation 'org.apache.logging.log4j:log4j-core:2.17.1'
    runtimeOnly 'org.apache.logging.log4j:log4j-slf4j18-impl:2.17.1'
    runtimeOnly 'com.amazonaws:aws-lambda-java-log4j2:1.5.1'
    ...
}
```

Lambda memuat file JAR dalam urutan alfabet Unicode. Jika beberapa file JAR ada di direktori `lib` berisi kelas yang sama, yang pertama digunakan. Anda dapat menggunakan skrip shell berikut untuk mengidentifikasi kelas duplikat:

Example test-zip.sh

```
mkdir -p expanded
unzip path/to/my/function.zip -d expanded
find ./expanded/lib -name '*.jar' | xargs -n1 zipinfo -1 | grep '.*.class' | sort |
 uniq -c | sort
```

Menggunakan lapisan untuk dependensi

Anda dapat mengemas dependensi fungsi Anda dalam lapisan untuk menjaga paket penerapan Anda tetap kecil dan mengelola dependensi secara independen. Untuk informasi selengkapnya, lihat the section called “Lapisan”.

Membangun paket deployment dengan Maven

Untuk membangun paket deployment dengan Maven, gunakan Plugin Maven Shade. Plugin membuat file JAR yang berisi kode fungsi terkompilasi dan semua dependensinya.

Example pom.xml – Konfigurasi plugin

```
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-shade-plugin</artifactId>
        <version>3.2.2</version>
        <configuration>
          <createDependencyReducedPom>false</createDependencyReducedPom>
        </configuration>
        <executions>
          <execution>
            <phase>package</phase>
            <goals>
              <goal>shade</goal>
            </goals>
          </execution>
        </executions>
      </plugin>
```

Untuk membuat paket deployment, gunakan perintah `mvn package`.

```
[INFO] Scanning for projects...
[INFO] -----------------------< com.example:java-maven >-----------------------
[INFO] Building java-maven-function 1.0-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
...
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ java-maven ---
[INFO] Building jar: target/java-maven-1.0-SNAPSHOT.jar
[INFO]
[INFO] --- maven-shade-plugin:3.2.2:shade (default) @ java-maven ---
[INFO] Including com.amazonaws:aws-lambda-java-core:jar:1.2.2 in the shaded jar.
[INFO] Including com.amazonaws:aws-lambda-java-events:jar:3.11.1 in the shaded jar.
[INFO] Including joda-time:joda-time:jar:2.6 in the shaded jar.
[INFO] Including com.google.code.gson:gson:jar:2.8.6 in the shaded jar.
[INFO] Replacing original artifact with shaded artifact.
[INFO] Replacing target/java-maven-1.0-SNAPSHOT.jar with target/java-maven-1.0-
SNAPSHOT-shaded.jar
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  8.321 s
[INFO] Finished at: 2020-03-03T09:07:19Z
[INFO] ------------------------------------------------------------------------
```

Perintah ini menghasilkan file JAR dalam direktori `target`.

> **Note**
>
> Jika Anda bekerja dengan JAR multi-rilis (MRJAR), Anda harus menyertakan MRJAR (yaitu JAR berbayang yang dihasilkan oleh plugin Maven Shade) di `lib` direktori dan zip sebelum mengunggah paket penerapan Anda ke Lambda. Jika tidak, Lambda mungkin tidak membongkar file JAR Anda dengan benar, menyebabkan `MANIFEST.MF` file Anda diabaikan.

Jika Anda menggunakan pustaka appender (`aws-lambda-java-log4j2`), Anda juga harus mengonfigurasi trafo untuk plugin Maven Shade. Pustaka trafo menggabungkan versi file cache yang muncul di pustaka appender dan Log4j.

Example pom.xml – Konfigurasi plugin dengan appender Log4j 2

```
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-shade-plugin</artifactId>
        <version>3.2.2</version>
        <configuration>
          <createDependencyReducedPom>false</createDependencyReducedPom>
        </configuration>
        <executions>
          <execution>
            <phase>package</phase>
            <goals>
              <goal>shade</goal>
            </goals>
            <configuration>
              <transformers>
                <transformer
implementation="com.github.edwgiz.maven_shade_plugin.log4j2_cache_transformer.PluginsCacheFile
                </transformer>
              </transformers>
            </configuration>
          </execution>
        </executions>
        <dependencies>
          <dependency>
            <groupId>com.github.edwgiz</groupId>
            <artifactId>maven-shade-plugin.log4j2-cachefile-transformer</artifactId>
```

```
            <version>2.13.0</version>
          </dependency>
        </dependencies>
      </plugin>
```

Mengunggah paket penerapan dengan konsol Lambda

Untuk membuat Fungsi Acara baru, Anda harus terlebih dahulu membuat fungsi di konsol, lalu mengunggah file.zip atau JAR Anda. Untuk memperbarui fungsi yang ada, buka halaman untuk fungsi Anda, lalu ikuti prosedur yang sama untuk menambahkan file.zip atau JAR Anda yang diperbarui.

Jika file paket penyebaran Anda kurang dari 50MB, Anda dapat membuat atau memperbarui fungsi dengan mengunggah file langsung dari mesin lokal Anda. Untuk file.zip atau JAR yang lebih besar dari 50MB, Anda harus mengunggah paket Anda ke bucket Amazon S3 terlebih dahulu. Untuk petunjuk tentang cara mengunggah file ke bucket Amazon S3 menggunakan Konsol Manajemen AWS, lihat Memulai Amazon S3. Untuk mengunggah file menggunakan AWS CLI, lihat Memindahkan objek di Panduan AWS CLI Pengguna.

> **Note**
>
> Anda tidak dapat mengubah jenis paket penerapan (.zip atau image kontainer) untuk fungsi yang ada. Misalnya, Anda tidak dapat mengonversi fungsi gambar kontainer untuk menggunakan arsip file.zip. Anda harus membuat fungsi baru.

Untuk membuat Fungsi Acara baru (konsol)

1. Buka halaman Fungsi Acara konsol Lambda dan pilih Buat Fungsi Acara.
2. Pilih Tulis dari awal.
3. Di bagian Informasi dasar, lakukan hal berikut:
    a. Untuk nama Fungsi, masukkan nama untuk fungsi Anda.
    b. Untuk Runtime, pilih runtime yang ingin Anda gunakan.
    c. (Opsional) Untuk Arsitektur, pilih arsitektur set instruksi untuk fungsi Anda. Arsitektur defaultnya adalah x86_64. Pastikan bahwa paket deployment .zip untuk fungsi Anda kompatibel dengan arsitektur set instruksi yang Anda pilih.

4. (Opsional) Di bagian Izin, luaskan Ubah peran eksekusi default. Anda dapat membuat peran Eksekusi baru atau menggunakan yang sudah ada.
5. Pilih Buat Fungsi Acara. Lambda menciptakan fungsi dasar 'Hello world' menggunakan runtime yang Anda pilih.

Untuk mengunggah arsip.zip atau JAR dari mesin lokal Anda (konsol)

1. Di halaman Fungsi Acara konsol Lambda, pilih fungsi yang ingin Anda unggah file.zip atau JAR.
2. Pilih tab Kode.
3. Di panel Sumber kode, pilih Unggah dari.
4. Pilih file.zip atau .jar.
5. Untuk mengunggah file.zip atau JAR, lakukan hal berikut:
   a. Pilih Unggah, lalu pilih file.zip atau JAR Anda di pemilih file.
   b. Pilih Buka.
   c. Pilih Simpan.

Untuk mengunggah arsip.zip atau JAR dari bucket Amazon S3 (konsol)

1. Di halaman Fungsi Acara konsol Lambda, pilih fungsi yang ingin Anda unggah file.zip atau JAR baru.
2. Pilih tab Kode.
3. Di panel Sumber kode, pilih Unggah dari.
4. Pilih lokasi Amazon S3.
5. Rekatkan URL tautan Amazon S3 dari file.zip Anda dan pilih Simpan.

Mengunggah paket penerapan dengan AWS CLI

Anda dapat menggunakan AWS CLIuntuk membuat fungsi baru atau memperbarui yang sudah ada menggunakan file.zip atau JAR. Gunakan perintah create-function dan update-function-code untuk menyebarkan paket.zip atau JAR Anda. Jika file Anda lebih kecil dari 50MB, Anda dapat mengunggah paket dari lokasi file di mesin build lokal Anda. Untuk file yang lebih besar, Anda harus mengunggah paket.zip atau JAR dari bucket Amazon S3. Untuk petunjuk tentang cara mengunggah file ke bucket Amazon S3 menggunakan AWS CLI, lihat Memindahkan objek di AWS CLI Panduan Pengguna.

> **Note**
>
> Jika Anda mengunggah file.zip atau JAR dari bucket Amazon S3 menggunakan AWS CLI bucket, bucket harus berada di lokasi Wilayah AWS yang sama dengan fungsi Anda.

Untuk membuat fungsi baru menggunakan file.zip atau JAR dengan AWS CLI, Anda harus menentukan yang berikut:

- Nama fungsi Anda (`--function-name`)
- Runtime () `--runtime` fungsi Anda
- Nama Sumber Daya Amazon (ARN) dari peran eksekusi fungsi Anda () `--role`
- Nama metode handler dalam kode fungsi Anda () `--handler`

Anda juga harus menentukan lokasi file.zip atau JAR Anda. Jika file.zip atau JAR Anda terletak di folder di mesin build lokal Anda, gunakan `--zip-file` opsi untuk menentukan jalur file, seperti yang ditunjukkan pada perintah contoh berikut.

```
aws lambda create-function --function-name myFunction \
--runtime java25 --handler example.handler \
--role arn:aws:iam::123456789012:role/service-role/my-lambda-role \
--zip-file fileb://myFunction.zip
```

Untuk menentukan lokasi file.zip di bucket Amazon S3, gunakan opsi seperti `--code` yang ditunjukkan pada perintah contoh berikut. Anda hanya perlu menggunakan `S3ObjectVersion` parameter untuk objek berversi.

```
aws lambda create-function --function-name myFunction \
--runtime java25 --handler example.handler \
--role arn:aws:iam::123456789012:role/service-role/my-lambda-role \
--code S3Bucket=amzn-s3-demo-
bucket,S3Key=myFileName.zip,S3ObjectVersion=myObjectVersion
```

Untuk memperbarui fungsi yang ada menggunakan CLI, Anda menentukan nama fungsi Anda menggunakan parameter. `--function-name` Anda juga harus menentukan lokasi file.zip yang ingin Anda gunakan untuk memperbarui kode fungsi Anda. Jika file.zip Anda terletak di folder di mesin build lokal Anda, gunakan `--zip-file` opsi untuk menentukan jalur file, seperti yang ditunjukkan pada perintah contoh berikut.

```
aws lambda update-function-code --function-name myFunction \
--zip-file fileb://myFunction.zip
```

Untuk menentukan lokasi file.zip di bucket Amazon S3, gunakan opsi `--s3-key` dan seperti `--s3-bucket` yang ditunjukkan pada perintah contoh berikut. Anda hanya perlu menggunakan `--s3-object-version` parameter untuk objek berversi.

```
aws lambda update-function-code --function-name myFunction \
--s3-bucket amzn-s3-demo-bucket --s3-key myFileName.zip --s3-object-version myObject
 Version
```

Mengunggah paket penerapan dengan AWS SAM

Anda dapat menggunakan AWS SAM untuk mengotomatiskan penerapan kode fungsi, konfigurasi, dan dependensi Anda. AWS SAM adalah ekstensi yang menyediakan sintaks yang disederhanakan untuk mendefinisikan aplikasi tanpa server. CloudFormation Templat contoh berikut mendefinisikan fungsi dengan paket deployment dalam direktori `build/distributions` yang digunakan Gradle:

Example template.yml

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: 'AWS::Serverless-2016-10-31'
Description: An AWS Lambda application that calls the Lambda API.
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: build/distributions/java-basic.zip
      Handler: example.Handler
      Runtime: java25
      Description: Java function
      MemorySize: 512
      Timeout: 10
      # Function's execution role
      Policies:
        - AWSLambdaBasicExecutionRole
        - AWSLambda_ReadOnlyAccess
        - AWSXrayWriteOnlyAccess
        - AWSLambdaVPCAccessExecutionRole
      Tracing: Active
```

Untuk membuat fungsi, gunakan perintah `package` dan `deploy`. Perintah ini adalah kustomisasi ke AWS CLI. Mereka membungkus perintah lain untuk mengunggah paket deployment ke Amazon S3, menulis ulang templat dengan objek URI, dan memperbarui kode fungsi.

Skrip contoh berikut menjalankan pembuatan Gradle dan mengunggah paket deployment yang dibuatnya. Ini menciptakan CloudFormation tumpukan saat pertama kali Anda menjalankannya. Jika tumpukan sudah ada, skrip akan memperbaruinya.

Example deploy.sh

```
#!/bin/bash
set -eo pipefail
aws cloudformation package --template-file template.yml --s3-bucket MY_BUCKET --output-
template-file out.yml
aws cloudformation deploy --template-file out.yml --stack-name java-basic --
capabilities CAPABILITY_NAMED_IAM
```

Untuk contoh kerja lengkap, lihat aplikasi sampel berikut:

Sampel aplikasi Lambda di Java

- example-java - Fungsi Acara Java yang menunjukkan bagaimana Anda dapat menggunakan Lambda untuk memproses pesanan. Fungsi ini menggambarkan cara mendefinisikan dan deserialisasi objek peristiwa input kustom, menggunakan AWS SDK, dan logging keluaran.
- java-basic - Kumpulan fungsi Java minimal dengan pengujian unit dan konfigurasi logging variabel.
- java-events - Kumpulan fungsi Java yang berisi kode kerangka untuk cara menangani peristiwa dari berbagai layanan seperti Amazon API Gateway, Amazon SQS, dan Amazon Kinesis. Fungsi-fungsi ini menggunakan versi terbaru dari pustaka aws-lambda-java-events (3.0.0 dan yang lebih baru). Contoh-contoh ini tidak memerlukan AWS SDK sebagai dependensi.
- s3-java – Fungsi Java yang memproses kejadian pemberitahuan dari Amazon S3 dan menggunakan Java Class Library (JCL) untuk membuat thumbnail dari file gambar yang diunggah.
- layer-java — Fungsi Java yang menggambarkan cara menggunakan lapisan Lambda untuk mengemas dependensi yang terpisah dari kode fungsi inti Anda.

Menyebarkan Fungsi Acara Lambda Java dengan gambar kontainer

Ada tiga cara untuk membangun image kontainer untuk Java Lambda Event Function:

- Menggunakan sebuah AWS gambar dasar untuk Java

  Gambar AWS dasar dimuat sebelumnya dengan runtime bahasa, klien antarmuka runtime untuk mengelola interaksi antara Lambda dan kode fungsi Anda, dan emulator antarmuka runtime untuk pengujian lokal.
- Menggunakan sebuah AWS OS-only gambar dasar

  AWS OS-only image dasar berisi distribusi Amazon Linux dan emulator antarmuka runtime. Gambar-gambar ini biasanya digunakan untuk membuat gambar kontainer untuk bahasa yang dikompilasi, seperti Go dan Rust, dan untuk versi bahasa atau bahasa yang Lambda tidak menyediakan gambar dasar, seperti Node.js 19. Anda juga dapat menggunakan gambar OS-only dasar untuk mengimplementasikan runtime kustom. Untuk membuat gambar kompatibel dengan Lambda, Anda harus menyertakan klien antarmuka runtime untuk Java dalam gambar.
- Menggunakan gambar AWS non-dasar

  Anda dapat menggunakan gambar dasar alternatif dari registri kontainer lain, seperti Alpine Linux atau Debian. Anda juga dapat menggunakan gambar kustom yang dibuat oleh organisasi Anda. Untuk membuat gambar kompatibel dengan Lambda, Anda harus menyertakan klien antarmuka runtime untuk Java dalam gambar.

> **Tip**
>
> Untuk mengurangi waktu yang dibutuhkan agar fungsi kontainer Lambda menjadi aktif, lihat Menggunakan build multi-tahap dalam dokumentasi Docker. Untuk membuat gambar kontainer yang efisien, ikuti Praktik terbaik untuk menulis Dockerfiles.

Halaman ini menjelaskan cara membuat, menguji, dan menyebarkan gambar kontainer untuk Lambda.

Topik

- AWS gambar dasar untuk Java
- Menggunakan sebuah AWS gambar dasar untuk Java

- Menggunakan gambar dasar alternatif dengan klien antarmuka runtime

AWS gambar dasar untuk Java

AWS menyediakan gambar dasar berikut untuk Java:

Tanda	Waktu berjalan	Sistem operasi	Dockerfile	penghentian
25	Jawa 25	Amazon Linux 2023	Dockerfile untuk Java 25 di GitHub	30 Jun 2029
21	Jawa 21	Amazon Linux 2023	Dockerfile untuk Java 21 di GitHub	30 Jun 2029
17.al20	Jawa 17	Amazon Linux 2023	Dockerfile untuk Java 17 di GitHub	30 Jun 2029
11.al20	Java 11	Amazon Linux 2023	Dockerfile untuk Java 11 di GitHub	30 Jun 2029
8.al202	Java 8	Amazon Linux 2023	Dockerfile untuk Java 8 di GitHub	30 Jun 2029
17	Jawa 17	Amazon Linux 2	Dockerfile untuk Java 17 di GitHub	30 Jun 2027
11	Java 11	Amazon Linux 2	Dockerfile untuk Java 11 di GitHub	30 Jun 2027
8.al2	Java 8	Amazon Linux 2	Dockerfile untuk Java 8 di GitHub	30 Jun 2027

Repositori Amazon ECR: gallery.ecr. aws/lambda/jawa

Gambar dasar Java 21 dan yang lebih baru didasarkan pada gambar kontainer minimal Amazon Linux 2023. Gambar dasar sebelumnya menggunakan Amazon Linux 2. AL2023 memberikan beberapa keunggulan dibandingkan Amazon Linux 2, termasuk jejak penyebaran yang lebih kecil dan versi pustaka yang diperbarui seperti. `glibc`

AL2023-based gambar menggunakan `microdnf` (symlinked as`dnf`) sebagai manajer paket, bukan`yum`, yang merupakan manajer paket default di Amazon Linux 2. `microdnf`adalah implementasi mandiri dari. `dnf` Untuk daftar paket yang disertakan dalam AL2023-based gambar, lihat kolom Penampung Minimal di Membandingkan paket yang diinstal di Amazon Linux 2023 Container Images. Untuk informasi selengkapnya tentang perbedaan antara AL2023 dan Amazon Linux 2, lihat Memperkenalkan AWS Lambda runtime Amazon Linux 2023 untuk di Blog Komputasi. AWS

> **Note**
>
> Untuk menjalankan AL2023-based gambar secara lokal, termasuk with AWS Serverless Application Model (AWS SAM), Anda harus menggunakan Docker versi 20.10.10 atau yang lebih baru.

Menggunakan sebuah AWS gambar dasar untuk Java

Prasyarat

Untuk menyelesaikan langkah-langkah di bagian ini, Anda harus memiliki hal-hal berikut:

- Java (misalnya, Amazon Corretto)
- Apache Maven atau Gradle
- AWS CLI versi 2
- Docker (versi minimum 25.0.0)
- Plugin buildx Docker.

Membuat gambar dari gambar dasar

Maven

1. Jalankan perintah berikut untuk membuat proyek Maven menggunakan pola dasar untuk Lambda. Parameter-parameter berikut diperlukan:

   - layanan — Layanan AWS Klien untuk digunakan dalam fungsi Lambda. Untuk daftar sumber yang tersedia, lihat aws-sdk-java - on. v2/services GitHub
   - wilayah — Wilayah AWS Tempat Anda ingin membuat fungsi Lambda.
   - GroupId — Namespace paket lengkap aplikasi Anda.

- ArtifactID — Nama proyek Anda. Ini menjadi nama direktori untuk proyek Anda.

Di Linux dan macOS, jalankan perintah ini:

```
mvn -B archetype:generate \
   -DarchetypeGroupId=software.amazon.awssdk \
   -DarchetypeArtifactId=archetype-lambda -Dservice=s3 -Dregion=US_WEST_2 \
   -DgroupId=com.example.myapp \
   -DartifactId=myapp
```

Di PowerShell, jalankan perintah ini:

```
mvn -B archetype:generate `
   "-DarchetypeGroupId=software.amazon.awssdk" `
   "-DarchetypeArtifactId=archetype-lambda" "-Dservice=s3" "-Dregion=US_WEST_2"
 `
   "-DgroupId=com.example.myapp" `
   "-DartifactId=myapp"
```

Pola dasar Maven untuk Lambda telah dikonfigurasi sebelumnya untuk dikompilasi dengan Java SE 8 dan menyertakan ketergantungan pada file. AWS SDK untuk Java Jika Anda membuat proyek Anda dengan pola dasar yang berbeda atau dengan menggunakan metode lain, Anda harus mengkonfigurasi compiler Java untuk Maven dan mendeklarasikan SDK sebagai dependensi.

2. Buka *myapp*`/src/main/java/com/example/`*myapp* direktori, dan temukan `App.java` filenya. Ini adalah kode untuk fungsi Lambda. Anda dapat menggunakan kode sampel yang disediakan untuk pengujian, atau menggantinya dengan kode Anda sendiri.
3. Arahkan kembali ke direktori root proyek, lalu buat Dockerfile baru dengan konfigurasi berikut:

   - Mengatur `FROM` properti ke URI dari gambar dasar.
   - Atur `CMD` argumen ke penangan fungsi Lambda.

   Perhatikan bahwa contoh Dockerfile tidak menyertakan instruksi USER. Saat Anda menyebarkan gambar kontainer ke Lambda, Lambda secara otomatis mendefinisikan pengguna Linux default dengan izin yang paling tidak memiliki hak istimewa. Ini berbeda dari perilaku Docker standar yang default ke `root` pengguna ketika tidak ada `USER` instruksi yang diberikan.

Example Dockerfile

```
FROM public.ecr.aws/lambda/java:21

# Copy function code and runtime dependencies from Maven layout
COPY target/classes ${LAMBDA_TASK_ROOT}
COPY target/dependency/* ${LAMBDA_TASK_ROOT}/lib/

# Set the CMD to your handler (could also be done as a parameter override
 outside of the Dockerfile)
CMD [ "com.example.myapp.App::handleRequest" ]
```

4. Kompilasi proyek dan kumpulkan dependensi runtime.

```
mvn compile dependency:copy-dependencies -DincludeScope=runtime
```

5. Buat image Docker dengan perintah docker build. Contoh berikut menamai gambar `docker-image` dan memberinya `test` tag. Untuk membuat gambar Anda kompatibel dengan Lambda, Anda harus menggunakan opsi. `--provenance=false`

```
docker buildx build --platform linux/amd64 --provenance=false -t docker-
image:test .
```

> **Note**
>
> Perintah menentukan `--platform linux/amd64` opsi untuk memastikan bahwa container Anda kompatibel dengan lingkungan eksekusi Lambda terlepas dari arsitektur mesin build Anda. Jika Anda berniat untuk membuat fungsi Lambda menggunakan arsitektur set instruksi ARM64, pastikan untuk mengubah perintah untuk menggunakan opsi sebagai `--platform linux/arm64` gantinya.

Gradle

1. Buat direktori untuk proyek, dan kemudian beralih ke direktori itu.

```
mkdir example
cd example
```

2. Jalankan perintah berikut agar Gradle membuat proyek aplikasi Java baru di direktori `example` di lingkungan Anda. Untuk Pilih skrip build DSL, pilih 2: Groovy.

```
gradle init --type java-application
```

3. Buka */example*/app/src/main/java/*example* direktori, dan temukan `App.java` filenya. Ini adalah kode untuk fungsi Lambda. Anda dapat menggunakan kode contoh berikut untuk pengujian, atau menggantinya dengan kode Anda sendiri.

   Example App.java

```
package com.example;
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
public class App implements RequestHandler<Object, String> {
    public String handleRequest(Object input, Context context) {
        return "Hello world!";
    }
}
```

4. Buka file `build.gradle`. Jika Anda menggunakan kode fungsi sampel dari langkah sebelumnya, ganti isinya `build.gradle` dengan yang berikut ini. Jika Anda menggunakan kode fungsi Anda sendiri, ubah `build.gradle` file Anda sesuai kebutuhan.

   Example build.gradle (Groovy DSL)

```
plugins {
  id 'java'
}
group 'com.example'
version '1.0-SNAPSHOT'
sourceCompatibility = 1.8
repositories {
  mavenCentral()
}
dependencies {
  implementation 'com.amazonaws:aws-lambda-java-core:1.2.1'
}
jar {
  manifest {
      attributes 'Main-Class': 'com.example.App'
  }
```

```
}
```

5. `gradle init`Perintah dari langkah 2 juga menghasilkan kasus uji dummy di `app/test` direktori. Untuk keperluan tutorial ini, lewati tes yang sedang berjalan dengan menghapus `/test` direktori.
6. Bangun proyek.

```
gradle build
```

7. Di direktori root proyek (`/example`), buat Dockerfile dengan konfigurasi berikut:

   - Mengatur `FROM` properti ke URI dari gambar dasar.
   - Gunakan perintah COPY untuk menyalin kode fungsi dan dependensi runtime ke`{LAMBDA_TASK_ROOT}`, variabel lingkungan. Lambda-defined
   - Atur `CMD` argumen ke penangan fungsi Lambda.

   Perhatikan bahwa contoh Dockerfile tidak menyertakan instruksi USER. Saat Anda menyebarkan gambar kontainer ke Lambda, Lambda secara otomatis mendefinisikan pengguna Linux default dengan izin yang paling tidak memiliki hak istimewa. Ini berbeda dari perilaku Docker standar yang default ke `root` pengguna ketika tidak ada USER instruksi yang diberikan.

   Example Dockerfile

```
FROM public.ecr.aws/lambda/java:21

# Copy function code and runtime dependencies from Gradle layout
COPY app/build/classes/java/main ${LAMBDA_TASK_ROOT}

# Set the CMD to your handler (could also be done as a parameter override
 outside of the Dockerfile)
CMD [ "com.example.App::handleRequest" ]
```

8. Buat image Docker dengan perintah docker build. Contoh berikut menamai gambar `docker-image` dan memberinya `test` tag. Untuk membuat gambar Anda kompatibel dengan Lambda, Anda harus menggunakan opsi. `--provenance=false`

```
docker buildx build --platform linux/amd64 --provenance=false -t docker-
image:test .
```

> **Note**
>
> Perintah menentukan `--platform linux/amd64` opsi untuk memastikan bahwa container Anda kompatibel dengan lingkungan eksekusi Lambda terlepas dari arsitektur mesin build Anda. Jika Anda berniat untuk membuat fungsi Lambda menggunakan arsitektur set instruksi ARM64, pastikan untuk mengubah perintah untuk menggunakan opsi sebagai `--platform linux/arm64` gantinya.

(Opsional) Uji gambar secara lokal

1. Mulai gambar Docker dengan perintah docker run. Dalam contoh ini, `docker-image` adalah nama gambar dan `test` tag.

   ```
   docker run --platform linux/amd64 -p 9000:8080 docker-image:test
   ```

   Perintah ini menjalankan gambar sebagai wadah dan membuat titik akhir lokal di`localhost:9000/2015-03-31/functions/function/invocations`.

   > **Note**
   >
   > Jika Anda membuat image Docker untuk arsitektur set instruksi ARM64, pastikan untuk menggunakan `--platform linux/arm64` opsi sebagai gantinya. `--platform linux/amd64`

2. Dari jendela terminal baru, posting acara ke titik akhir lokal.

   Linux/macOS

   Di Linux dan macOS, jalankan perintah berikut: `curl`

   ```
   curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
   ```

   Perintah ini memanggil fungsi dengan peristiwa kosong dan mengembalikan respons. Jika Anda menggunakan kode fungsi Anda sendiri daripada kode fungsi sampel, Anda mungkin ingin memanggil fungsi dengan payload JSON. Contoh:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d
 '{"payload":"hello world!"}'
```

PowerShell

Dalam PowerShell, jalankan `Invoke-WebRequest` perintah berikut:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
invocations" -Method Post -Body '{}' -ContentType "application/json"
```

Perintah ini memanggil fungsi dengan peristiwa kosong dan mengembalikan respons. Jika Anda menggunakan kode fungsi Anda sendiri daripada kode fungsi sampel, Anda mungkin ingin memanggil fungsi dengan payload JSON. Contoh:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
invocations" -Method Post -Body '{"payload":"hello world!"}' -ContentType
 "application/json"
```

3. Dapatkan ID kontainer.

```
docker ps
```

4. Gunakan perintah docker kill untuk menghentikan kontainer. Dalam perintah ini, ganti `3766c4ab331c` dengan ID kontainer dari langkah sebelumnya.

```
docker kill 3766c4ab331c
```

Menyebarkan gambar

Untuk mengunggah gambar ke Amazon ECR dan membuat fungsi Lambda

1. Jalankan perintah get-login-password untuk mengautentikasi CLI Docker ke registri Amazon ECR Anda.
   - Tetapkan `--region` nilai ke Wilayah AWS tempat Anda ingin membuat repositori Amazon ECR.
   - Ganti `111122223333` dengan Akun AWS ID Anda.

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --
password-stdin 111122223333.dkr.ecr.us-east-1.amazonaws.com
```

2. Buat repositori di Amazon ECR menggunakan perintah create-repository.

```
aws ecr create-repository --repository-name hello-world --region us-east-1 --image-
scanning-configuration scanOnPush=true --image-tag-mutability MUTABLE
```

> **Note**
>
> Repositori Amazon ECR harus sama Wilayah AWS dengan fungsi Lambda.

Jika berhasil, Anda melihat respons seperti ini:

```
{
    "repository": {
        "repositoryArn": "arn:aws:ecr:us-east-1:111122223333:repository/hello-
world",
        "registryId": "111122223333",
        "repositoryName": "hello-world",
        "repositoryUri": "111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-
world",
        "createdAt": "2023-03-09T10:39:01+00:00",
        "imageTagMutability": "MUTABLE",
        "imageScanningConfiguration": {
            "scanOnPush": true
        },
        "encryptionConfiguration": {
            "encryptionType": "AES256"
        }
    }
}
```

3. Salin `repositoryUri` dari output pada langkah sebelumnya.
4. Jalankan perintah tag docker untuk menandai gambar lokal Anda ke repositori Amazon ECR Anda sebagai versi terbaru. Dalam perintah ini:

- `docker-image:test`adalah nama dan tag gambar Docker Anda. Ini adalah nama gambar dan tag yang Anda tentukan dalam `docker build` perintah.
- Ganti `<ECRrepositoryUri>` dengan `repositoryUri` yang Anda salin. Pastikan untuk menyertakan `:latest` di akhir URI.

```
docker tag docker-image:test <ECRrepositoryUri>:latest
```

Contoh:

```
docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

5. Jalankan perintah docker push untuk menyebarkan gambar lokal Anda ke repositori Amazon ECR. Pastikan untuk menyertakan `:latest` di akhir URI repositori.

```
docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

6. Buat peran eksekusi untuk fungsi tersebut, jika Anda belum memilikinya. Anda memerlukan Nama Sumber Daya Amazon (ARN) dari peran tersebut di langkah berikutnya.
7. Buat fungsi Lambda. Untuk`ImageUri`, tentukan URI repositori dari sebelumnya. Pastikan untuk menyertakan `:latest` di akhir URI.

```
aws lambda create-function \
  --function-name hello-world \
  --package-type Image \
  --code ImageUri=111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
  --role arn:aws:iam::111122223333:role/lambda-ex
```

> **Note**
>
> Anda dapat membuat fungsi menggunakan gambar di AWS akun yang berbeda, selama gambar berada di Wilayah yang sama dengan fungsi Lambda. Untuk informasi selengkapnya, lihat Izin lintas akun Amazon ECR.

8. Memanggil fungsi.

```
aws lambda invoke --function-name hello-world response.json
```

Anda akan melihat tanggapan seperti ini:

```
{
  "ExecutedVersion": "$LATEST",
  "StatusCode": 200
}
```

9. Untuk melihat output dari fungsi, periksa `response.json` file.

Untuk memperbarui kode fungsi, Anda harus membuat gambar lagi, mengunggah gambar baru ke repositori Amazon ECR, dan kemudian menggunakan perintah update-function-code untuk menyebarkan gambar ke fungsi Lambda.

Lambda menyelesaikan tag gambar ke intisari gambar tertentu. Ini berarti bahwa jika Anda mengarahkan tag gambar yang digunakan untuk menyebarkan fungsi ke gambar baru di Amazon ECR, Lambda tidak secara otomatis memperbarui fungsi untuk menggunakan gambar baru.

Untuk menyebarkan gambar baru ke fungsi Lambda yang sama, Anda harus menggunakan perintah update-function-code, meskipun tag gambar di Amazon ECR tetap sama. Dalam contoh berikut, `--publish` opsi membuat versi baru dari fungsi menggunakan gambar kontainer yang diperbarui.

```
aws lambda update-function-code \
  --function-name hello-world \
  --image-uri 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
  --publish
```

Menggunakan gambar dasar alternatif dengan klien antarmuka runtime

Jika Anda menggunakan gambar OS-only dasar atau gambar dasar alternatif, Anda harus menyertakan klien antarmuka runtime dalam gambar Anda. Klien antarmuka runtime memperluasAPI runtime, yang mengelola interaksi antara Lambda dan kode fungsi Anda.

Instal klien antarmuka runtime untuk Java di Dockerfile Anda, atau sebagai ketergantungan dalam proyek Anda. Misalnya, untuk menginstal klien antarmuka runtime menggunakan manajer paket Maven, tambahkan yang berikut ini ke file Anda: `pom.xml`

```
<dependency>
```

```
    <groupId>com.amazonaws</groupId>
    <artifactId>aws-lambda-java-runtime-interface-client</artifactId>
    <version>2.3.2</version>
</dependency>
```

Untuk detail paket, lihat AWS Lambda Java Runtime Interface Client di Maven Central Repository. Anda juga dapat meninjau kode sumber klien antarmuka runtime di repositori AWS Lambda Java Support Libraries GitHub .

Contoh berikut menunjukkan cara membangun image container untuk Java menggunakan image Amazon Corretto. Amazon Corretto adalah distribusi tanpa biaya, multiplatform, siap produksi dari Open Java Development Kit (OpenJDK). Proyek Maven menyertakan klien antarmuka runtime sebagai dependensi.

Prasyarat

Untuk menyelesaikan langkah-langkah di bagian ini, Anda harus memiliki hal-hal berikut:

- Java (misalnya, Amazon Corretto)
- Apache Maven
- AWS CLI versi 2
- Docker (versi minimum 25.0.0)
- Plugin buildx Docker.

Membuat gambar dari gambar dasar alternatif

1. Buat proyek Maven. Parameter-parameter berikut diperlukan:

   - GroupId — Namespace paket lengkap aplikasi Anda.
   - ArtifactID — Nama proyek Anda. Ini menjadi nama direktori untuk proyek Anda.

   Linux/macOS

   ```
   mvn -B archetype:generate \
      -DarchetypeArtifactId=maven-archetype-quickstart \
      -DgroupId=example \
      -DartifactId=myapp \
      -DinteractiveMode=false
   ```

PowerShell

```
mvn -B archetype:generate `
   -DarchetypeArtifactId=maven-archetype-quickstart `
   -DgroupId=example `
   -DartifactId=myapp `
   -DinteractiveMode=false
```

2. Buka direktori proyek.

```
cd myapp
```

3. Buka `pom.xml` file dan ganti isinya dengan yang berikut ini. File ini menyertakan aws-lambda-java-runtime-interface-client sebagai dependensi. Atau, Anda dapat menginstal klien antarmuka runtime di Dockerfile. Namun, pendekatan paling sederhana adalah memasukkan perpustakaan sebagai ketergantungan.

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/
maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>example</groupId>
  <artifactId>hello-lambda</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>hello-lambda</name>
  <url>http://maven.apache.org</url>
  <properties>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
  </properties>
  <dependencies>
    <dependency>
      <groupId>com.amazonaws</groupId>
      <artifactId>aws-lambda-java-runtime-interface-client</artifactId>
      <version>2.3.2</version>
    </dependency>
  </dependencies>
  <build>
    <plugins>
      <plugin>
```

```
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-dependency-plugin</artifactId>
        <version>3.1.2</version>
        <executions>
          <execution>
            <id>copy-dependencies</id>
            <phase>package</phase>
            <goals>
              <goal>copy-dependencies</goal>
            </goals>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>
</project>
```

4. Buka *myapp*`/src/main/java/com/example/`*myapp* direktori, dan temukan `App.java` filenya. Ini adalah kode untuk fungsi Lambda. Ganti kode dengan yang berikut ini.

   Example fungsi handler

   ```
   package example;

   public class App {
       public static String sayHello() {
           return "Hello world!";
       }
   }
   ```

5. `mvn -B archetype:generate`Perintah dari langkah 1 juga menghasilkan kasus uji dummy di `src/test` direktori. Untuk keperluan tutorial ini, lewati tes yang sedang berjalan dengan menghapus seluruh `/test` direktori yang dihasilkan ini.
6. Arahkan kembali ke direktori root proyek, lalu buat Dockerfile baru. Contoh berikut Dockerfile menggunakan gambar Amazon Corretto. Amazon Corretto adalah distribusi OpenJDK tanpa biaya, multiplatform, dan siap produksi.

   - Atur properti `FROM` untuk URI dari gambar dasar.
   - Atur `ENTRYPOINT` ke modul yang Anda inginkan untuk menjalankan wadah Docker saat dimulai. Dalam hal ini, modul adalah klien antarmuka runtime.
   - Atur `CMD` argumen ke penangan fungsi Lambda.

Perhatikan bahwa contoh Dockerfile tidak menyertakan instruksi USER. Saat Anda menyebarkan gambar kontainer ke Lambda, Lambda secara otomatis mendefinisikan pengguna Linux default dengan izin yang paling tidak memiliki hak istimewa. Ini berbeda dari perilaku Docker standar yang default ke `root` pengguna ketika tidak ada USER instruksi yang diberikan.

Example Dockerfile

```
FROM public.ecr.aws/amazoncorretto/amazoncorretto:21 as base

# Configure the build environment
FROM base as build
RUN yum install -y maven
WORKDIR /src

# Cache and copy dependencies
ADD pom.xml .
RUN mvn dependency:go-offline dependency:copy-dependencies

# Compile the function
ADD . .
RUN mvn package

# Copy the function artifact and dependencies onto a clean base
FROM base
WORKDIR /function

COPY --from=build /src/target/dependency/*.jar ./
COPY --from=build /src/target/*.jar ./

# Set runtime interface client as default command for the container runtime
ENTRYPOINT [ "/usr/bin/java", "-cp", "./*",
 "com.amazonaws.services.lambda.runtime.api.client.AWSLambda" ]
# Pass the name of the function handler as an argument to the runtime
CMD [ "example.App::sayHello" ]
```

7. Buat image Docker dengan perintah docker build. Contoh berikut menamai gambar `docker-image` dan memberinya `test` tag. Untuk membuat gambar Anda kompatibel dengan Lambda, Anda harus menggunakan opsi. `--provenance=false`

```
docker buildx build --platform linux/amd64 --provenance=false -t docker-image:test
 .
```

> **Note**
>
> Perintah menentukan `--platform linux/amd64` opsi untuk memastikan bahwa container Anda kompatibel dengan lingkungan eksekusi Lambda terlepas dari arsitektur mesin build Anda. Jika Anda berniat untuk membuat fungsi Lambda menggunakan arsitektur set instruksi ARM64, pastikan untuk mengubah perintah untuk menggunakan opsi sebagai `--platform linux/arm64` gantinya.

(Opsional) Uji gambar secara lokal

Gunakan emulator antarmuka runtime untuk menguji gambar secara lokal. Anda dapat membangun emulator ke dalam gambar Anda atau menggunakan prosedur berikut untuk menginstalnya di mesin lokal Anda.

Untuk menginstal dan menjalankan emulator antarmuka runtime di mesin lokal Anda

1. Dari direktori proyek Anda, jalankan perintah berikut untuk mengunduh emulator antarmuka runtime (arsitektur x86-64) dari GitHub dan menginstalnya di mesin lokal Anda.

   Linux/macOS

   ```
   mkdir -p ~/.aws-lambda-rie && \
       curl -Lo ~/.aws-lambda-rie/aws-lambda-rie https://github.com/aws/aws-lambda-
   runtime-interface-emulator/releases/latest/download/aws-lambda-rie && \
       chmod +x ~/.aws-lambda-rie/aws-lambda-rie
   ```

   Untuk menginstal emulator arm64, ganti URL GitHub repositori di perintah sebelumnya dengan yang berikut:

   ```
   https://github.com/aws/aws-lambda-runtime-interface-emulator/releases/latest/
   download/aws-lambda-rie-arm64
   ```

PowerShell

```
$dirPath = "$HOME\.aws-lambda-rie"
if (-not (Test-Path $dirPath)) {
    New-Item -Path $dirPath -ItemType Directory
}

$downloadLink = "https://github.com/aws/aws-lambda-runtime-interface-emulator/
releases/latest/download/aws-lambda-rie"
$destinationPath = "$HOME\.aws-lambda-rie\aws-lambda-rie"
Invoke-WebRequest -Uri $downloadLink -OutFile $destinationPath
```

Untuk menginstal emulator arm64, ganti `$downloadLink` dengan yang berikut ini:

```
https://github.com/aws/aws-lambda-runtime-interface-emulator/releases/latest/
download/aws-lambda-rie-arm64
```

2. Mulai gambar Docker dengan perintah docker run. Perhatikan hal-hal berikut:

   - `docker-image`adalah nama gambar dan `test` tag.
   - `/usr/bin/java -cp './*' com.amazonaws.services.lambda.runtime.api.client.AWSLambda example.App::sayHello`adalah `ENTRYPOINT` diikuti oleh `CMD` dari Dockerfile Anda.

   Linux/macOS

   ```
   docker run --platform linux/amd64 -d -v ~/.aws-lambda-rie:/aws-lambda -p
    9000:8080 \
       --entrypoint /aws-lambda/aws-lambda-rie \
       docker-image:test \
           /usr/bin/java -cp './*'
    com.amazonaws.services.lambda.runtime.api.client.AWSLambda
    example.App::sayHello
   ```

   PowerShell

   ```
   docker run --platform linux/amd64 -d -v "$HOME\.aws-lambda-rie:/aws-lambda" -p
    9000:8080 `
   --entrypoint /aws-lambda/aws-lambda-rie `
   ```

```
docker-image:test `
    /usr/bin/java -cp './*'
 com.amazonaws.services.lambda.runtime.api.client.AWSLambda
 example.App::sayHello
```

Perintah ini menjalankan gambar sebagai wadah dan membuat titik akhir lokal di`localhost:9000/2015-03-31/functions/function/invocations`.

> **Note**
>
> Jika Anda membuat image Docker untuk arsitektur set instruksi ARM64, pastikan untuk menggunakan `--platform linux/arm64` opsi sebagai gantinya. `--platform linux/amd64`

3. Posting acara ke titik akhir lokal.

   Linux/macOS

   Di Linux dan macOS, jalankan perintah berikut: `curl`

   ```
   curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
   ```

   Perintah ini memanggil fungsi dengan peristiwa kosong dan mengembalikan respons. Jika Anda menggunakan kode fungsi Anda sendiri daripada kode fungsi sampel, Anda mungkin ingin memanggil fungsi dengan payload JSON. Contoh:

   ```
   curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d
    '{"payload":"hello world!"}'
   ```

   PowerShell

   Dalam PowerShell, jalankan `Invoke-WebRequest` perintah berikut:

   ```
   Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
   invocations" -Method Post -Body '{}' -ContentType "application/json"
   ```

Perintah ini memanggil fungsi dengan peristiwa kosong dan mengembalikan respons. Jika Anda menggunakan kode fungsi Anda sendiri daripada kode fungsi sampel, Anda mungkin ingin memanggil fungsi dengan payload JSON. Contoh:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
invocations" -Method Post -Body '{"payload":"hello world!"}' -ContentType
 "application/json"
```

4. Dapatkan ID kontainer.

```
docker ps
```

5. Gunakan perintah docker kill untuk menghentikan kontainer. Dalam perintah ini, ganti `3766c4ab331c` dengan ID kontainer dari langkah sebelumnya.

```
docker kill 3766c4ab331c
```

Menyebarkan gambar

Untuk mengunggah gambar ke Amazon ECR dan membuat fungsi Lambda

1. Jalankan perintah get-login-password untuk mengautentikasi CLI Docker ke registri Amazon ECR Anda.
   - Tetapkan `--region` nilai ke Wilayah AWS tempat Anda ingin membuat repositori Amazon ECR.
   - Ganti `111122223333` dengan Akun AWS ID Anda.

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --
password-stdin 111122223333.dkr.ecr.us-east-1.amazonaws.com
```

2. Buat repositori di Amazon ECR menggunakan perintah create-repository.

```
aws ecr create-repository --repository-name hello-world --region us-east-1 --image-
scanning-configuration scanOnPush=true --image-tag-mutability MUTABLE
```

> **Note**
>
> Repositori Amazon ECR harus sama Wilayah AWS dengan fungsi Lambda.

Jika berhasil, Anda melihat respons seperti ini:

```
{
    "repository": {
        "repositoryArn": "arn:aws:ecr:us-east-1:111122223333:repository/hello-
world",
        "registryId": "111122223333",
        "repositoryName": "hello-world",
        "repositoryUri": "111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-
world",
        "createdAt": "2023-03-09T10:39:01+00:00",
        "imageTagMutability": "MUTABLE",
        "imageScanningConfiguration": {
            "scanOnPush": true
        },
        "encryptionConfiguration": {
            "encryptionType": "AES256"
        }
    }
}
```

3. Salin `repositoryUri` dari output pada langkah sebelumnya.
4. Jalankan perintah tag docker untuk menandai gambar lokal Anda ke repositori Amazon ECR Anda sebagai versi terbaru. Dalam perintah ini:

   - `docker-image:test`adalah nama dan tag gambar Docker Anda. Ini adalah nama gambar dan tag yang Anda tentukan dalam `docker build` perintah.
   - Ganti <ECRrepositoryUri> dengan `repositoryUri` yang Anda salin. Pastikan untuk menyertakan `:latest` di akhir URI.

```
docker tag docker-image:test <ECRrepositoryUri>:latest
```

Contoh:

```
docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

5. Jalankan perintah docker push untuk menyebarkan gambar lokal Anda ke repositori Amazon ECR. Pastikan untuk menyertakan `:latest` di akhir URI repositori.

```
docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

6. Buat peran eksekusi untuk fungsi tersebut, jika Anda belum memilikinya. Anda memerlukan Nama Sumber Daya Amazon (ARN) dari peran tersebut di langkah berikutnya.
7. Buat fungsi Lambda. Untuk`ImageUri`, tentukan URI repositori dari sebelumnya. Pastikan untuk menyertakan `:latest` di akhir URI.

```
aws lambda create-function \
  --function-name hello-world \
  --package-type Image \
  --code ImageUri=111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
  --role arn:aws:iam::111122223333:role/lambda-ex
```

> **Note**
>
> Anda dapat membuat fungsi menggunakan gambar di AWS akun yang berbeda, selama gambar berada di Wilayah yang sama dengan fungsi Lambda. Untuk informasi selengkapnya, lihat Izin lintas akun Amazon ECR.

8. Memanggil fungsi.

```
aws lambda invoke --function-name hello-world response.json
```

Anda akan melihat tanggapan seperti ini:

```
{
  "ExecutedVersion": "$LATEST",
  "StatusCode": 200
}
```

9. Untuk melihat output dari fungsi, periksa `response.json` file.

Untuk memperbarui kode fungsi, Anda harus membuat gambar lagi, mengunggah gambar baru ke repositori Amazon ECR, dan kemudian menggunakan perintah update-function-code untuk menyebarkan gambar ke fungsi Lambda.

Lambda menyelesaikan tag gambar ke intisari gambar tertentu. Ini berarti bahwa jika Anda mengarahkan tag gambar yang digunakan untuk menyebarkan fungsi ke gambar baru di Amazon ECR, Lambda tidak secara otomatis memperbarui fungsi untuk menggunakan gambar baru.

Untuk menyebarkan gambar baru ke fungsi Lambda yang sama, Anda harus menggunakan perintah update-function-code, meskipun tag gambar di Amazon ECR tetap sama. Dalam contoh berikut, `--publish` opsi membuat versi baru dari fungsi menggunakan gambar kontainer yang diperbarui.

```
aws lambda update-function-code \
  --function-name hello-world \
  --image-uri 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
  --publish
```

Bekerja dengan lapisan untuk Fungsi Acara Java Lambda

Gunakan lapisan Lambda untuk mengemas kode dan dependensi yang ingin Anda gunakan kembali di beberapa fungsi. Lapisan biasanya berisi dependensi pustaka, runtime kustom, atau file konfigurasi. Membuat layer melibatkan tiga langkah umum:

1. Package konten layer Anda. Ini berarti membuat arsip file.zip yang berisi dependensi yang ingin Anda gunakan dalam fungsi Anda.
2. Buat layer di Lambda.
3. Tambahkan layer ke fungsi Anda.

Topik

- Package konten layer Anda
- Buat layer di Lambda
- Tambahkan layer ke fungsi Anda

Package konten layer Anda

Untuk membuat layer, bundel paket Anda ke dalam arsip file.zip yang memenuhi persyaratan berikut:

- Pastikan bahwa versi Java yang mengacu pada Maven atau Gradle sama dengan versi Java dari fungsi yang ingin Anda gunakan. Misalnya, untuk fungsi Java 25, `mvn -v` perintah harus mencantumkan Java 25 di output.
- Dependensi Anda harus disimpan di `java/lib` direktori, di root file.zip. Untuk informasi selengkapnya, lihat Jalur lapisan untuk setiap runtime Lambda.
- Paket di lapisan Anda harus kompatibel dengan Linux. Fungsi Lambda berjalan di Amazon Linux.

Anda dapat membuat lapisan yang berisi pustaka Java pihak ketiga atau modul dan paket Java Anda sendiri. Prosedur berikut menggunakan Maven. Anda juga dapat menggunakan Gradle untuk mengemas konten layer Anda.

Untuk membuat layer menggunakan dependensi Maven

1. Buat proyek Apache Maven dengan `pom.xml` file yang mendefinisikan dependensi Anda.

Contoh berikut termasuk Jackson Databind untuk pemrosesan JSON. <build>Bagian ini menggunakan maven-dependency-plugin untuk membuat file JAR terpisah untuk setiap dependensi alih-alih menggabungkannya ke dalam satu uber-jar. Jika Anda ingin membuat uber-jar, gunakan maven-shade-plugin.

Example pom.xml

```
<dependencies>
    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-databind</artifactId>
        <version>2.17.0</version>
    </dependency>
</dependencies>

<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.13.0</version>
            <configuration>
                <source>21</source>
                <target>21</target>
                <release>21</release>
            </configuration>
        </plugin>

        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-dependency-plugin</artifactId>
            <version>3.6.1</version>
            <executions>
                <execution>
                    <id>copy-dependencies</id>
                    <phase>package</phase>
                    <goals>
                        <goal>copy-dependencies</goal>
                    </goals>
                    <configuration>
                        <outputDirectory>${project.build.directory}/lib</
outputDirectory>
```

```
                    </configuration>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>
```

2. Bangun proyek. Perintah ini menciptakan semua file JAR ketergantungan dalam `target/lib/` direktori.

```
mvn clean package
```

3. Buat struktur direktori yang diperlukan untuk layer Anda:

```
mkdir -p java/lib
```

4. Salin file JAR ketergantungan ke `java/lib` direktori:

```
cp target/lib/*.jar java/lib/
```

5. Zip konten lapisan:

   Linux/macOS

   ```
   zip -r layer.zip java/
   ```

   PowerShell

   ```
   Compress-Archive -Path .\java -DestinationPath .\layer.zip
   ```

   Struktur direktori file.zip Anda akan terlihat seperti ini:

```
java/
### lib/
    ### jackson-databind-2.17.0.jar
    ### jackson-core-2.17.0.jar
    ### jackson-annotations-2.17.0.jar
```

> **Note**
>
> Pastikan file.zip Anda menyertakan `java` direktori di tingkat root dengan `lib` di dalamnya. Struktur ini memastikan bahwa Lambda dapat menemukan dan mengimpor pustaka Anda. Setiap dependensi disimpan sebagai file JAR terpisah daripada dibundel ke dalam uber-jar.

Buat layer di Lambda

Anda dapat mempublikasikan layer Anda menggunakan konsol AWS CLI atau Lambda.

AWS CLI

Jalankan AWS CLI perintah publish-layer-version untuk membuat layer Lambda:

```
aws lambda publish-layer-version --layer-name my-layer --zip-file fileb://layer.zip
 --compatible-runtimes java25
```

Parameter runtime yang kompatibel adalah opsional. Saat ditentukan, Lambda menggunakan parameter ini untuk memfilter lapisan di konsol Lambda.

Console

Untuk membuat lapisan (konsol)

1. Buka halaman Fungsi di konsol Lambda.
2. Pilih Buat lapisan.
3. Pilih Unggah file.zip, lalu unggah arsip.zip yang Anda buat sebelumnya.
4. (Opsional) Untuk runtime yang kompatibel, pilih runtime Java yang sesuai dengan versi Java yang Anda gunakan untuk membangun layer Anda.
5. Pilih Buat.

Tambahkan layer ke fungsi Anda

AWS CLI

Untuk melampirkan layer ke fungsi Anda, jalankan perintah AWS CLI update-function-configuration. Untuk `--layers` parameter, gunakan lapisan ARN. ARN harus menentukan versi

(misalnya,`arn:aws:lambda:us-east-1:123456789012:layer:my-layer:1`). Untuk informasi selengkapnya, lihat Versi lapisan dan lapisan.

```
aws lambda update-function-configuration --function-name my-function --cli-binary-format raw-in-base64-out --layers "arn:aws:lambda:us-east-1:123456789012:layer:my-layer:1"
```

cli-binary-formatOpsi ini diperlukan jika Anda menggunakan AWS CLI versi 2. Untuk menjadikan ini pengaturan default, jalankan`aws configure set cli-binary-format raw-in-base64-out`. Untuk informasi selengkapnya, lihat opsi baris perintah global yang AWS CLI didukung di Panduan AWS Command Line Interface Pengguna untuk Versi 2.

Console

Untuk menambahkan lapisan ke fungsi

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih fungsinya.
3. Gulir ke bawah ke bagian Layers, lalu pilih Add a layer.
4. Di bawah Pilih layer, pilih Custom layers, dan kemudian pilih layer Anda.

> **Note**
>
> Jika Anda tidak menambahkan runtime yang kompatibel saat membuat layer, layer Anda tidak akan tercantum di sini. Anda dapat menentukan lapisan ARN sebagai gantinya.

5. Pilih Tambahkan.

Kustomisasi serialisasi untuk Lambda Java Event Functions

Runtime terkelola Lambda Java mendukung serialisasi khusus untuk acara JSON. Serialisasi khusus dapat menyederhanakan kode Anda dan berpotensi meningkatkan kinerja.

Topik

- Kapan menggunakan serialisasi khusus
- Menerapkan serialisasi kustom
- Menguji serialisasi kustom

Kapan menggunakan serialisasi khusus

Ketika fungsi Lambda Anda dipanggil, data peristiwa input perlu dideserialisasi ke objek Java, dan output dari fungsi Anda perlu diserialisasikan kembali ke format yang dapat dikembalikan sebagai respons fungsi. Runtime terkelola Lambda Java menyediakan kemampuan serialisasi dan deserialisasi default yang berfungsi dengan baik untuk menangani muatan peristiwa dari berbagai layanan, AWS seperti Amazon API Gateway dan Amazon Simple Queue Service (Amazon SQS). Untuk bekerja dengan peristiwa integrasi layanan ini dalam fungsi Anda, tambahkan dependensi aws-java-lambda-events ke project Anda. AWS Pustaka ini berisi objek Java yang mewakili peristiwa integrasi layanan ini.

Anda juga dapat menggunakan objek Anda sendiri untuk mewakili acara JSON yang Anda berikan ke fungsi Lambda Anda. Runtime terkelola mencoba membuat serial JSON ke instance baru objek Anda dengan perilaku defaultnya. Jika serializer default tidak memiliki perilaku yang diinginkan untuk kasus penggunaan Anda, gunakan serialisasi khusus.

Misalnya, asumsikan bahwa penangan fungsi Anda mengharapkan `Vehicle` kelas sebagai input, dengan struktur berikut:

```
public class Vehicle {
    private String vehicleType;
    private long vehicleId;
}
```

Namun, payload acara JSON terlihat seperti ini:

```
{
    "vehicle-type": "car",
```

```
    "vehicleID": 123
}
```

Dalam skenario ini, serialisasi default dalam runtime terkelola mengharapkan nama properti JSON cocok dengan nama properti kelas Java case unta (,). `vehicleType` `vehicleId` Karena nama properti dalam acara JSON tidak dalam kasus unta (`vehicle-type`,`vehicleID`), Anda harus menggunakan serialisasi khusus.

Menerapkan serialisasi kustom

Gunakan Antarmuka Penyedia Layanan untuk memuat serializer pilihan Anda alih-alih logika serialisasi default runtime yang dikelola. Anda dapat membuat serial muatan acara JSON Anda langsung ke objek Java, menggunakan antarmuka standar. `RequestHandler`

Untuk menggunakan serialisasi kustom di Java Lambda Event Function Anda

1. Tambahkan pustaka aws-lambda-java-core sebagai dependensi. Pustaka ini mencakup CustomPojoSerializerantarmuka, bersama dengan definisi antarmuka lain untuk bekerja dengan Java di Lambda.
2. Buat file bernama `com.amazonaws.services.lambda.runtime.CustomPojoSerializer` di `src/main/resources/META-INF/services/` direktori proyek Anda.
3. Dalam file ini, tentukan nama yang sepenuhnya memenuhi syarat dari implementasi serializer kustom Anda, yang harus mengimplementasikan antarmuka. `CustomPojoSerializer` Contoh:

   ```
   com.mycompany.vehicles.CustomLambdaSerialzer
   ```
4. Terapkan `CustomPojoSerializer` antarmuka untuk memberikan logika serialisasi kustom Anda.
5. Gunakan `RequestHandler` antarmuka standar dalam fungsi Lambda Anda. Runtime terkelola akan menggunakan serializer kustom Anda.

Untuk contoh lebih lanjut tentang cara menerapkan serialisasi kustom menggunakan pustaka populer seperti FastJSON, Gson, Moshi, dan jackson-jr, lihat contoh serialisasi kustom di repositori. AWS GitHub

Menguji serialisasi kustom

Uji fungsi Anda untuk memastikan bahwa logika serialisasi dan deserialisasi Anda berfungsi seperti yang diharapkan. Anda dapat menggunakan AWS Serverless Application Model Command Line

Interface (AWS SAMCLI) untuk meniru pemanggilan payload Lambda Anda. Ini dapat membantu Anda dengan cepat menguji dan mengulangi fungsi Anda saat Anda memperkenalkan serializer khusus.

1. Buat file dengan payload acara JSON yang ingin Anda panggil fungsi Anda lalu panggil file. AWS SAMCLI
2. Jalankan perintah sam local invoke untuk memanggil fungsi Anda secara lokal. Contoh:

```
sam local invoke -e src/test/resources/event.json
```

Untuk informasi selengkapnya, lihat Memanggil fungsi Lambda secara lokal dengan. AWS SAM

Kustomisasi perilaku startup runtime Java untuk Fungsi Acara Lambda

Halaman ini menjelaskan pengaturan khusus untuk Java Event Functions di AWS Lambda. Anda dapat menggunakan pengaturan ini untuk menyesuaikan perilaku startup runtime Java. Ini dapat mengurangi latensi fungsi secara keseluruhan dan meningkatkan kinerja fungsi secara keseluruhan, tanpa harus memodifikasi kode apa pun.

Bagian-bagian

- Memahami variabel lingkungan JAVA_TOOL_OPTIONS
- Patch Log4j untuk Log4Shell
- Ahead-of-Time (AOT) dan cache CDS

Memahami variabel lingkungan **`JAVA_TOOL_OPTIONS`**

Di Java, Lambda mendukung variabel `JAVA_TOOL_OPTIONS` lingkungan untuk mengatur variabel baris perintah tambahan di Lambda. Anda dapat menggunakan variabel lingkungan ini dengan berbagai cara, seperti untuk menyesuaikan pengaturan kompilasi berjenjang. Contoh berikutnya menunjukkan bagaimana menggunakan variabel `JAVA_TOOL_OPTIONS` lingkungan untuk kasus penggunaan ini.

Contoh: Menyesuaikan pengaturan kompilasi berjenjang

Kompilasi berjenjang adalah fitur dari mesin virtual Java (JVM). Anda dapat menggunakan pengaturan kompilasi berjenjang tertentu untuk memanfaatkan kompiler just-in-time (JIT) JVM dengan sebaik-baiknya. Biasanya, kompiler C1 dioptimalkan untuk waktu start-up yang cepat. Kompiler C2 dioptimalkan untuk kinerja keseluruhan terbaik, tetapi juga menggunakan lebih banyak memori dan membutuhkan waktu lebih lama untuk mencapainya.

Ada 5 tingkat kompilasi berjenjang yang berbeda. Pada Level 0, JVM menafsirkan kode byte Java. Pada Level 4, JVM menggunakan kompiler C2 untuk menganalisis data profil yang dikumpulkan selama startup aplikasi. Seiring waktu, ia memantau penggunaan kode untuk mengidentifikasi pengoptimalan terbaik.

Menyesuaikan tingkat kompilasi berjenjang dapat membantu Anda menyetel kinerja fungsi Java Anda. Untuk fungsi kecil yang dijalankan dengan cepat, menyetel kompilasi berjenjang ke level 1 dapat membantu meningkatkan kinerja start dingin dengan meminta JVM menggunakan kompiler C1. Pengaturan ini dengan cepat menghasilkan kode asli yang dioptimalkan tetapi tidak menghasilkan data profil apa pun dan tidak pernah menggunakan kompiler C2. Untuk fungsi yang lebih besar dan

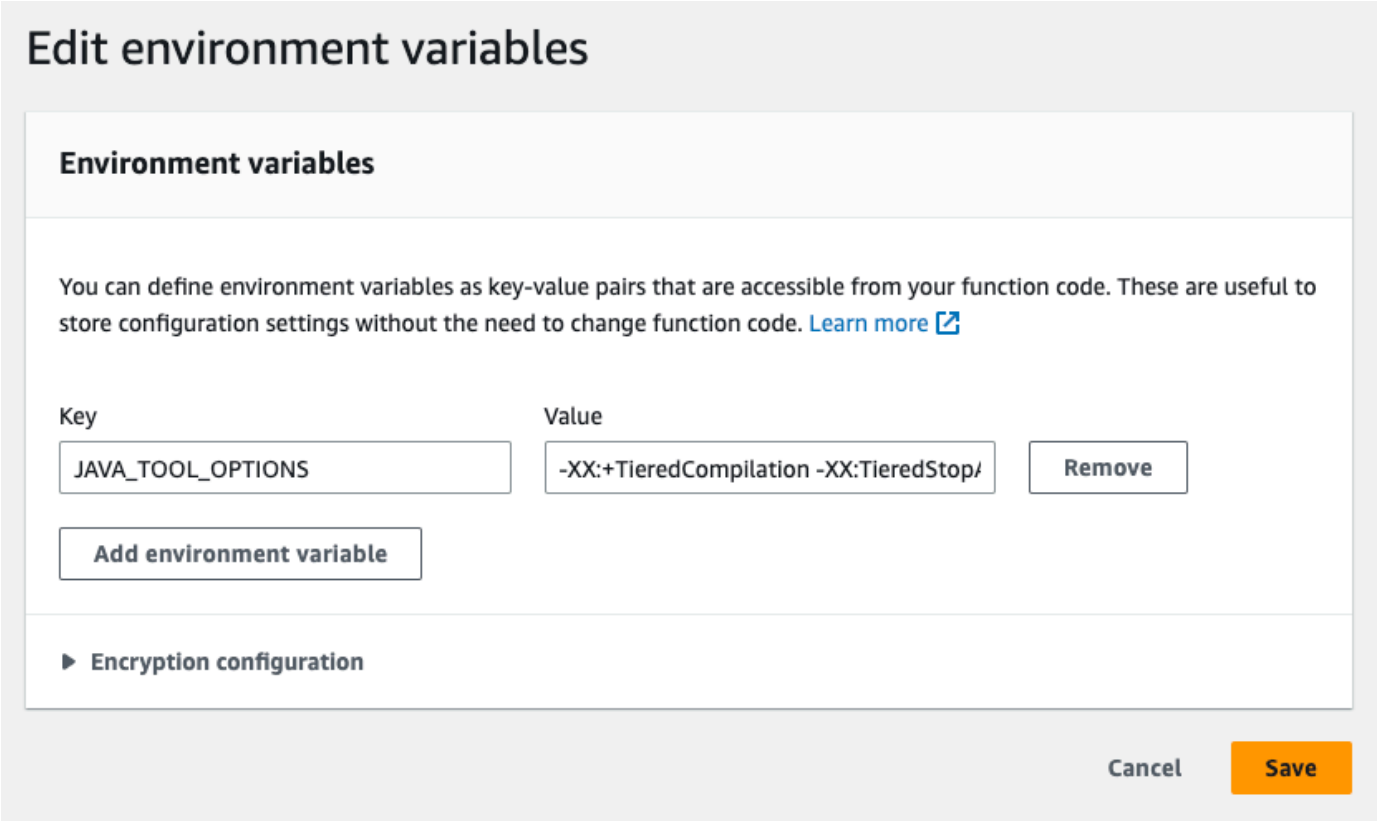
intensif komputasi, pengaturan kompilasi berjenjang ke level 4 memaksimalkan kinerja keseluruhan dengan mengorbankan konsumsi memori tambahan dan pekerjaan pengoptimalan tambahan selama pemanggilan pertama setelah setiap lingkungan eksekusi Lambda disediakan.

Untuk runtime Java 11 dan di bawahnya, Lambda menggunakan pengaturan kompilasi berjenjang JVM default. Untuk Java 17 dan Java 21, Lambda mengonfigurasi JVM untuk menghentikan kompilasi berjenjang di level 1 secara default. Dari Java 25, Lambda masih menghentikan kompilasi berjenjang di level 1 secara default, kecuali saat menggunakan SnapStart atau Konkurensi yang disediakan, dalam hal ini pengaturan JVM default digunakan.

Ini meningkatkan kinerja untuk SnapStart dan Konkurensi yang disediakan tanpa menimbulkan penalti start dingin karena kompilasi berjenjang dilakukan di luar jalur pemanggilan dalam kasus ini. Untuk memaksimalkan manfaat ini, Anda dapat menggunakan priming - mengeksekusi jalur kode selama inisialisasi fungsi untuk memicu JIT sebelum mengambil SnapStart snapshot atau ketika lingkungan eksekusi Konkurensi yang Disediakan telah disediakan sebelumnya. Untuk informasi lebih lanjut, lihat posting blog Mengoptimalkan kinerja start dingin AWS Lambda menggunakan strategi priming lanjutan dengan. SnapStart

Untuk menyesuaikan pengaturan kompilasi berjenjang (konsol)

1. Buka halaman Fungsi Acara di konsol Lambda.
2. Pilih fungsi Java yang ingin Anda sesuaikan kompilasi berjenjang.
3. Pilih tab Konfigurasi, lalu pilih variabel Lingkungan di menu sebelah kiri.
4. Pilih Edit.
5. Pilih Tambahkan variabel lingkungan.
6. Untuk kuncinya, masukkan`JAVA_TOOL_OPTIONS`. Untuk nilainya, masukkan`-XX:+TieredCompilation -XX:TieredStopAtLevel=1`.



7. Pilih Simpan.

> **Note**
>
> Anda juga dapat menggunakan Lambda SnapStart untuk mengurangi masalah start dingin. SnapStartmenggunakan snapshot cache dari lingkungan eksekusi Anda untuk meningkatkan kinerja start-up secara signifikan. Untuk informasi selengkapnya tentang SnapStart fitur, batasan, dan wilayah yang didukung, lihatMeningkatkan kinerja startup dengan Lambda SnapStart.

Contoh: Menyesuaikan perilaku GC menggunakan JAVA_TOOL_OPTIONS

Java 11 runtime menggunakan Serial garbage collector (GC) untuk pengumpulan sampah. Secara default, runtime Java 17 juga menggunakan Serial GC. Namun, dengan Java 17 Anda juga dapat menggunakan variabel `JAVA_TOOL_OPTIONS` lingkungan untuk mengubah GC default. Anda dapat memilih antara Parallel GC dan Shenandoah GC.

Misalnya, jika beban kerja Anda menggunakan lebih banyak memori dan beberapa CPU, pertimbangkan untuk menggunakan Parallel GC untuk kinerja yang lebih baik. Anda dapat melakukan ini dengan menambahkan berikut ini ke nilai variabel `JAVA_TOOL_OPTIONS` lingkungan Anda:

```
-XX:+UseParallelGC
```

Jika beban kerja Anda memiliki banyak objek berumur pendek, Anda dapat mengambil manfaat dari konsumsi memori yang lebih rendah dengan mengaktifkan mode generasi pengumpul sampah Shenandoah yang diperkenalkan di Jawa 25. Untuk melakukan ini, tambahkan yang berikut ini ke nilai variabel `JAVA_TOOL_OPTIONS` lingkungan Anda:

```
-XX:+UseShenandoahGC -XX:ShenandoahGCMode=generational
```

Patch Log4j untuk Log4Shell

Runtime Lambda untuk Java 8, 11, 17, dan 21 menyertakan tambalan untuk mengurangi kerentanan Log4Shell (CVE-2021-44228) di Log4j, kerangka kerja logging Java yang populer. Patch ini menimbulkan overhead kinerja start dingin. Jika Anda menggunakan versi Log4j yang ditambal (versi 2.17.0 atau yang lebih baru), Anda dapat menonaktifkan tambalan ini untuk meningkatkan kinerja start dingin. Untuk menonaktifkan tambalan, atur variabel `AWS_LAMBDA_DISABLE_CVE_2021_44228_PROTECTION` lingkungan `ketrue`.

Mulai dari Java 25, runtime Lambda tidak lagi menyertakan patch Log4Shell. Anda harus memverifikasi bahwa Anda menggunakan Log4j versi 2.17.0 atau yang lebih baru.

Ahead-of-Time (AOT) dan cache CDS

Dimulai dengan Java 25, runtime Lambda menyertakan cache Ahead-of-Time (AOT) untuk klien antarmuka runtime Java (RIC), komponen runtime yang secara aktif melakukan polling untuk peristiwa dari Lambda Runtime API. Ini meningkatkan kinerja start dingin.

Cache AOT khusus untuk build JVM. Saat Lambda memperbarui runtime terkelola, Lambda juga memperbarui cache AOT untuk RIC. Namun, jika Anda menerapkan cache AOT Anda sendiri, cache ini mungkin tidak valid atau mengakibatkan perilaku tak terduga setelah pembaruan runtime. Oleh karena itu, kami sangat menyarankan untuk tidak menggunakan cache AOT saat menggunakan runtime terkelola. Untuk menggunakan cache AOT, Anda harus menerapkan fungsi Anda menggunakan gambar kontainer.

Cache AOT tidak dapat digunakan dengan cache Berbagi Data Kelas (CDS). Jika Anda menerapkan cache CDS di fungsi Lambda Anda, maka Lambda menonaktifkan cache AOT.

Menggunakan objek konteks Lambda untuk mengambil informasi Java Event Function

Saat Lambda menjalankan fungsi Anda, ia meneruskan objek konteks ke handler. Objek ini menyediakan metode dan properti yang memberikan informasi tentang lingkungan invokasi, fungsi, dan eksekusi.

Metode konteks

- `getRemainingTimeInMillis()` – Mengembalikan jumlah milidetik yang tersisa sebelum waktu eksekusi habis.
- `getFunctionName()` – Mengembalikan nama fungsi Lambda.
- `getFunctionVersion()` – Mengembalikan versi fungsi.
- `getInvokedFunctionArn()` – Mengembalikan Amazon Resource Name (ARN) yang digunakan untuk mengaktifkan fungsi. Menunjukkan jika pemicu menyebutkan nomor versi atau alias.
- `getMemoryLimitInMB()` – Mengembalikan jumlah memori yang dialokasikan untuk fungsi tersebut.
- `getAwsRequestId()` – Mengembalikan pengidentifikasi permintaan invokasi.
- `getLogGroupName()` – Mengembalikan grup log untuk fungsi.
- `getLogStreamName()` – Mengembalikan aliran log untuk instans fungsi.
- `getIdentity()` – (aplikasi seluler) Mengembalikan informasi tentang Amazon Cognito yang mengesahkan permintaan.
- `getClientContext()` – (aplikasi seluler) Mengembalikan konteks klien yang disediakan untuk Lambda oleh aplikasi klien.
- `getLogger()` – Mengembalikan objek logger untuk fungsi.

Contoh berikut menunjukkan fungsi yang menggunakan objek konteks untuk mengakses logger Lambda.

Example Handler.java

```
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.LambdaLogger;
import com.amazonaws.services.lambda.runtime.RequestHandler;
```

```
import java.util.Map;

// Handler value: example.Handler
public class Handler implements RequestHandler<Map<String,String>, Void>{

  @Override
  public Void handleRequest(Map<String,String> event, Context context)
  {
    LambdaLogger logger = context.getLogger();
    logger.log("EVENT TYPE: " + event.getClass());
    return null;
  }
}
```

Fungsi mencatat jenis kelas dari acara yang masuk sebelum kembali`null`.

Example output log

```
EVENT TYPE: class java.util.LinkedHashMap
```

Antarmuka untuk objek konteks tersedia di pustaka aws-lambda-java-core. Anda dapat menerapkan antarmuka ini untuk membuat kelas konteks untuk pengujian. Contoh berikut menunjukkan kelas konteks yang mengembalikan nilai tiruan untuk sebagian besar properti dan logger uji kerja.

Example src/test/java/example/TestContext.java

```
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.CognitoIdentity;
import com.amazonaws.services.lambda.runtime.ClientContext;
import com.amazonaws.services.lambda.runtime.LambdaLogger;

public class TestContext implements Context{

  public TestContext() {}
  public String getAwsRequestId(){
    return new String("495b12a8-xmpl-4eca-8168-160484189f99");
  }
  public String getLogGroupName(){
    return new String("/aws/lambda/my-function");
  }
```

```
  public String getLogStreamName(){
    return new String("2020/02/26/[$LATEST]704f8dxmpla04097b9134246b8438f1a");
  }
  public String getFunctionName(){
    return new String("my-function");
  }
  public String getFunctionVersion(){
    return new String("$LATEST");
  }
  public String getInvokedFunctionArn(){
    return new String("arn:aws:lambda:us-east-2:123456789012:function:my-function");
  }
  public CognitoIdentity getIdentity(){
    return null;
  }
  public ClientContext getClientContext(){
    return null;
  }
  public int getRemainingTimeInMillis(){
    return 300000;
  }
  public int getMemoryLimitInMB(){
    return 512;
  }
  public LambdaLogger getLogger(){
    return new TestLogger();
  }

}
```

Untuk informasi lebih lanjut tentang log, lihat Log dan pantau Fungsi Acara Java Lambda.

Konteks dalam aplikasi sampel

GitHub Repositori untuk panduan ini mencakup contoh aplikasi yang menunjukkan penggunaan objek konteks. Setiap contoh aplikasi menyertakan skrip untuk penyebaran dan pembersihan yang mudah, templat AWS Serverless Application Model (AWS SAM), dan sumber daya pendukung.

Sampel aplikasi Lambda di Java

- example-java - Fungsi Acara Java yang menunjukkan bagaimana Anda dapat menggunakan Lambda untuk memproses pesanan. Fungsi ini menggambarkan cara mendefinisikan dan deserialisasi objek peristiwa masukan kustom, menggunakan AWS SDK, dan logging keluaran.

- java-basic - Kumpulan fungsi Java minimal dengan pengujian unit dan konfigurasi logging variabel.
- java-events - Kumpulan fungsi Java yang berisi kode kerangka untuk cara menangani peristiwa dari berbagai layanan seperti Amazon API Gateway, Amazon SQS, dan Amazon Kinesis. Fungsi-fungsi ini menggunakan versi terbaru dari pustaka aws-lambda-java-events (3.0.0 dan yang lebih baru). Contoh-contoh ini tidak memerlukan AWS SDK sebagai dependensi.
- s3-java – Fungsi Java yang memproses kejadian pemberitahuan dari Amazon S3 dan menggunakan Java Class Library (JCL) untuk membuat thumbnail dari file gambar yang diunggah.
- layer-java — Fungsi Java yang menggambarkan cara menggunakan lapisan Lambda untuk mengemas dependensi yang terpisah dari kode fungsi inti Anda.

Log dan pantau Fungsi Acara Java Lambda

AWS Lambda secara otomatis memonitor fungsi Lambda dan mengirim entri log ke Amazon. CloudWatch Fungsi Lambda Anda dilengkapi dengan grup CloudWatch log Log dan aliran log untuk setiap instance fungsi Anda. Lingkungan runtime Lambda mengirimkan detail tentang setiap pemanggilan dan output lainnya dari kode fungsi Anda ke aliran log. Untuk informasi selengkapnya tentang CloudWatch Log, lihatMengirim log Fungsi Peristiwa Lambda ke Log CloudWatch.

Untuk mengeluarkan log dari kode fungsi Anda, Anda dapat menggunakan metode di java.lang.System, atau modul logging apa pun yang menulis ke stdout atau stderr.

Bagian-bagian

- Membuat fungsi yang mengembalikan log
- Menggunakan kontrol logging lanjutan Lambda dengan Java
- Menerapkan logging lanjutan dengan Log4j2 dan SLF4J
- Menggunakan alat logging dan pustaka lainnya
- Menggunakan Powertools untuk AWS Lambda (Java) dan AWS SAM untuk logging terstruktur
- Melihat log di konsol Lambda
- Melihat log di CloudWatch konsol
- Melihat log menggunakan AWS Command Line Interface (AWS CLI)
- Menghapus log
- Kode pencatatan sampel

Membuat fungsi yang mengembalikan log

Untuk menghasilkan log dari kode fungsi, Anda dapat menggunakan metode di java.lang.System, atau modul pencatatan apa pun yang menulis ke `stdout` atau `stderr`. Pustaka aws-lambda-java-core menyediakan kelas logger bernama `LambdaLogger` yang dapat Anda akses dari objek konteks. Kelas logger mendukung log beberapa baris.

Contoh berikut menggunakan logger `LambdaLogger` yang disediakan oleh objek konteks.

Example Handler.java

```
// Handler value: example.Handler
public class Handler implements RequestHandler<Object, String>{
  Gson gson = new GsonBuilder().setPrettyPrinting().create();
```

```
  @Override
  public String handleRequest(Object event, Context context)
  {
    LambdaLogger logger = context.getLogger();
    String response = new String("SUCCESS");
    // log execution details
    logger.log("ENVIRONMENT VARIABLES: " + gson.toJson(System.getenv()));
    logger.log("CONTEXT: " + gson.toJson(context));
    // process event
    logger.log("EVENT: " + gson.toJson(event));
    return response;
  }
}
```

Example format log

```
START RequestId: 6bc28136-xmpl-4365-b021-0ce6b2e64ab0 Version: $LATEST
ENVIRONMENT VARIABLES:
{
    "_HANDLER": "example.Handler",
    "AWS_EXECUTION_ENV": "AWS_Lambda_java8",
    "AWS_LAMBDA_FUNCTION_MEMORY_SIZE": "512",
    ...
}
CONTEXT:
{
    "memoryLimit": 512,
    "awsRequestId": "6bc28136-xmpl-4365-b021-0ce6b2e64ab0",
    "functionName": "java-console",
    ...
}
EVENT:
{
  "records": [
    {
      "messageId": "19dd0b57-xmpl-4ac1-bd88-01bbb068cb78",
      "receiptHandle": "MessageReceiptHandle",
      "body": "Hello from SQS!",
        ...
    }
  ]
}
END RequestId: 6bc28136-xmpl-4365-b021-0ce6b2e64ab0
```

```
REPORT RequestId: 6bc28136-xmpl-4365-b021-0ce6b2e64ab0 Duration: 198.50 ms Billed
 Duration: 724 ms Memory Size: 512 MB Max Memory Used: 90 MB Init Duration: 524.75 ms
```

Runtime Java mencatat baris `START`, `END`, dan `REPORT` untuk setiap invokasi. Baris laporan memberikan detail berikut:

Laporkan bidang data baris

- RequestId – ID permintaan unik untuk invokasi.
- Durasi – Jumlah waktu yang digunakan oleh metode handler fungsi Anda gunakan untuk memproses peristiwa.
- Durasi yang Ditagih – Jumlah waktu yang ditagihkan untuk invokasi.
- Ukuran Memori – Jumlah memori yang dialokasikan untuk fungsi.
- Memori Maks yang Digunakan – Jumlah memori yang digunakan oleh fungsi. Saat pemanggilan berbagi lingkungan eksekusi, Lambda melaporkan memori maksimum yang digunakan di semua pemanggilan. Perilaku ini dapat menghasilkan nilai yang dilaporkan lebih tinggi dari yang diharapkan.
- Durasi Init – Untuk permintaan pertama yang dilayani, lama waktu yang diperlukan runtime untuk memuat fungsi dan menjalankan kode di luar metode handler.
- XRAY TraceId — Untuk permintaan yang dilacak, ID AWS X-Ray jejak.
- SegmentId— Untuk permintaan yang dilacak, ID X-Ray segmen.
- Diambil Sampel – Untuk permintaan yang dilacak, hasil pengambilan sampel.

Menggunakan kontrol logging lanjutan Lambda dengan Java

Untuk memberi Anda kontrol lebih besar atas bagaimana log fungsi Anda ditangkap, diproses, dan dikonsumsi, Anda dapat mengonfigurasi opsi logging berikut untuk runtime Java yang didukung:

- Format log - pilih antara teks biasa dan format JSON terstruktur untuk log fungsi Anda
- Tingkat log - untuk log dalam format JSON, pilih tingkat detail log yang dikirim CloudWatch Lambda, seperti ERROR, DEBUG, atau INFO
- Grup log - pilih grup CloudWatch log fungsi Anda mengirim log ke

Untuk informasi selengkapnya tentang opsi pencatatan ini, dan petunjuk tentang cara mengonfigurasi fungsi Anda untuk menggunakannya, lihatthe section called “Mengkonfigurasi kontrol logging lanjutan untuk Lambda Event Functions”.

Untuk menggunakan format log dan opsi tingkat log dengan fungsi Java Lambda Anda, lihat panduan di bagian berikut.

Menggunakan format log JSON terstruktur dengan Java

Jika Anda memilih JSON untuk format log fungsi Anda, Lambda mengirimkan output log menggunakan kelas sebagai JSON `LambdaLogger` terstruktur. Setiap objek log JSON berisi setidaknya empat pasangan nilai kunci dengan kunci berikut:

- `"timestamp"`- waktu pesan log dihasilkan
- `"level"`- tingkat log yang ditetapkan untuk pesan
- `"message"`- isi pesan log
- `"AWSrequestId"`- ID permintaan unik untuk pemanggilan fungsi

Bergantung pada metode logging yang Anda gunakan, output log dari fungsi Anda yang ditangkap dalam format JSON juga dapat berisi pasangan nilai kunci tambahan.

Untuk menetapkan level ke log yang Anda buat menggunakan `LambdaLogger` logger, Anda perlu memberikan `LogLevel` argumen dalam perintah logging Anda seperti yang ditunjukkan pada contoh berikut.

Example Kode logging Java

```
LambdaLogger logger = context.getLogger();
logger.log("This is a debug log", LogLevel.DEBUG);
```

Output log ini dengan kode contoh ini akan ditangkap di CloudWatch Log sebagai berikut:

Example Catatan log JSON

```
{
    "timestamp":"2023-11-01T00:21:51.358Z",
    "level":"DEBUG",
    "message":"This is a debug log",
    "AWSrequestId":"93f25699-2cbf-4976-8f94-336a0aa98c6f"
}
```

Jika Anda tidak menetapkan level ke output log Anda, Lambda secara otomatis menetapkannya INFO level.

Jika kode Anda sudah menggunakan pustaka logging lain untuk menghasilkan log terstruktur JSON, Anda tidak perlu membuat perubahan apa pun. Lambda tidak menyandikan dua kali log apa pun yang sudah dikodekan JSON. Bahkan jika Anda mengonfigurasi fungsi Anda untuk menggunakan format log JSON, output logging Anda muncul CloudWatch dalam struktur JSON yang Anda tentukan.

Menggunakan penyaringan tingkat log dengan Java

AWS Lambda Untuk memfilter log aplikasi Anda sesuai dengan tingkat lognya, fungsi Anda harus menggunakan log berformat JSON. Anda dapat mencapai ini dengan dua cara:

- Buat output log menggunakan standar `LambdaLogger` dan konfigurasikan fungsi Anda untuk menggunakan pemformatan log JSON. Lambda kemudian memfilter output log Anda menggunakan pasangan nilai kunci "level" di objek JSON yang dijelaskan di. the section called "Menggunakan format log JSON terstruktur dengan Java" Untuk mempelajari cara mengonfigurasi format log fungsi Anda, lihatthe section called "Mengkonfigurasi kontrol logging lanjutan untuk Lambda Event Functions".
- Gunakan pustaka atau metode logging lain untuk membuat log terstruktur JSON dalam kode Anda yang menyertakan pasangan nilai kunci "level" yang menentukan tingkat keluaran log. Anda dapat menggunakan pustaka logging apa pun yang dapat menulis log JSON ke `stdout` ataustderr. Misalnya, Anda dapat menggunakan Powertools for AWS Lambda atau paket Log4j2 untuk menghasilkan output log terstruktur JSON dari kode Anda. Lihat the section called "Menggunakan Powertools untuk AWS Lambda (Java) dan AWS SAM untuk logging terstruktur" dan the section called "Menerapkan logging lanjutan dengan Log4j2 dan SLF4J" untuk mempelajari lebih lanjut.

Ketika Anda mengonfigurasi fungsi Anda untuk menggunakan pemfilteran tingkat log, Anda harus memilih dari opsi berikut untuk tingkat log yang ingin Lambda kirim ke Log: CloudWatch

Tingkat log	Penggunaan standar
TRACE (paling detail)	Informasi paling halus yang digunakan untuk melacak jalur eksekusi kode Anda
DEBUG	Informasi terperinci untuk debugging sistem
INFO	Pesan yang merekam operasi normal fungsi Anda

Tingkat log	Penggunaan standar
WARN	Pesan tentang potensi kesalahan yang dapat menyebabkan perilaku tak terduga jika tidak ditangani
Kesalahan	Pesan tentang masalah yang mencegah kode berfungsi seperti yang diharapkan
FATAL (paling detail)	Pesan tentang kesalahan serius yang menyebabkan aplikasi berhenti berfungsi

Agar Lambda dapat memfilter log fungsi Anda, Anda juga harus menyertakan pasangan nilai `"timestamp"` kunci dalam keluaran log JSON Anda. Waktu harus ditentukan dalam format stempel waktu RFC 3339 yang valid. Jika Anda tidak menyediakan stempel waktu yang valid, Lambda menetapkan log INFO level dan menambahkan stempel waktu untuk Anda.

Lambda mengirimkan log dari level yang dipilih dan lebih rendah ke. CloudWatch Misalnya, jika Anda mengonfigurasi level log WARN, Lambda mengirimkan log yang sesuai dengan level WARN, ERROR, dan FATAL.

Menerapkan logging lanjutan dengan Log4j2 dan SLF4J

> **Note**
>
> AWS Lambda tidak menyertakan Log4j2 dalam runtime terkelola atau gambar wadah dasarnya. Oleh karena itu, ini tidak terpengaruh oleh masalah yang dijelaskan dalam CVE-2021-44228, CVE-2021-45046, dan CVE-2021-45105.
> Untuk kasus di mana fungsi pelanggan menyertakan versi Log4j2 yang terpengaruh, kami telah menerapkan perubahan pada runtime terkelola Lambda Java dan gambar wadah dasar yang membantu mengurangi masalah di,, dan. CVE-2021-44228 CVE-2021-45046 CVE-2021-45105 Sebagai hasil dari perubahan ini, pelanggan yang menggunakan Log4J2 mungkin melihat entri log tambahan, mirip dengan "". `Transforming org/apache/logging/log4j/core/lookup/JndiLookup (java.net.URLClassLoader@...)` String log apa pun yang mereferensikan jndi mapper di output Log4J2 diganti dengan "". `Patched JndiLookup::lookup()`
> Terlepas dari perubahan ini, kami sangat mendorong semua pelanggan yang fungsinya termasuk Log4j2 untuk memperbarui ke versi terbaru. Secara khusus, pelanggan yang

menggunakan pustaka aws-lambda-java-log4j2 dalam fungsinya harus memperbarui ke versi 1.5.0 (atau yang lebih baru), dan menerapkan ulang fungsinya. Versi ini memperbarui dependensi utilitas Log4j2 yang mendasarinya ke versi 2.17.0 (atau yang lebih baru). Biner aws-lambda-java-log4j2 yang diperbarui tersedia di repositori Maven dan kode sumbernya tersedia di Github.

Terakhir, perhatikan bahwa pustaka apa pun yang terkait dengan aws-lambda-java-log4j (v1.0.0 atau 1.0.1) tidak boleh digunakan dalam keadaan apa pun. Pustaka ini terkait dengan versi 1.x dari log4j yang berakhir pada tahun 2015. Pustaka tidak didukung, tidak dipelihara, tidak ditambal, dan memiliki kerentanan keamanan yang diketahui.

Untuk menyesuaikan output log, mendukung pencatatan selama pengujian unit, dan log panggilan AWS SDK, gunakan Apache Log4j2 dengan SLF4J. Log4j adalah perpustakaan logging untuk program Java yang dapat Anda gunakan untuk mengkonfigurasi tingkat log dan menggunakan pustaka appender. SLF4J adalah pustaka fasad yang dapat Anda gunakan untuk mengubah perpustakaan mana yang Anda gunakan tanpa mengubah kode fungsi Anda.

Untuk menambahkan ID permintaan ke log fungsi Anda, gunakan appender di pustaka aws-lambda-java-log4j2.

Example src/main/resources/log4j2.xml — Konfigurasi lampiran

```
<Configuration>
  <Appenders>
    <Lambda name="Lambda" format="${env:AWS_LAMBDA_LOG_FORMAT:-TEXT}">
       <LambdaTextFormat>
         <PatternLayout>
             <pattern>%d{yyyy-MM-dd HH:mm:ss} %X{AWSRequestId} %-5p %c{1} - %m%n </
pattern>
         </PatternLayout>
       </LambdaTextFormat>
       <LambdaJSONFormat>
         <JsonTemplateLayout eventTemplateUri="classpath:LambdaLayout.json" />
       </LambdaJSONFormat>
     </Lambda>
   </Appenders>
   <Loggers>
     <Root level="${env:AWS_LAMBDA_LOG_LEVEL:-INFO}">
       <AppenderRef ref="Lambda"/>
     </Root>
     <Logger name="software.amazon.awssdk" level="WARN" />
```

```
    <Logger name="software.amazon.awssdk.request" level="DEBUG" />
  </Loggers>
</Configuration>
```

Anda dapat memutuskan bagaimana log Log4j2 Anda dikonfigurasi untuk teks biasa atau output JSON dengan menentukan tata letak di bawah tag dan. `<LambdaTextFormat>` `<LambdaJSONFormat>`

Dalam contoh ini, dalam mode teks, setiap baris ditambahkan dengan tanggal, waktu, ID permintaan, tingkat log, dan nama kelas. Dalam mode JSON, `<JsonTemplateLayout>` digunakan dengan konfigurasi yang dikirimkan bersama dengan `aws-lambda-java-log4j2` perpustakaan.

SLF4J adalah pustaka fasad untuk login ke kode Java. Dalam kode fungsi Anda, Anda menggunakan pabrik logger SLF4J untuk mengambil logger dengan metode untuk tingkat log seperti `info()` dan `warn()`. Dalam konfigurasi build Anda, Anda menyertakan pustaka pencatatan dan adaptor SLF4J di classpath. Dengan mengubah pustaka dalam konfigurasi build, Anda dapat mengubah tipe logger tanpa mengubah kode fungsi Anda. SLF4J diperlukan untuk menangkap log dari SDK for Java.

Dalam contoh kode berikut, kelas handler menggunakan SLF4J untuk mengambil logger.

Example src/main/java/example/HandlerS3.java — Logging dengan SLF4J

```
package example;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.S3Event;

import static org.apache.logging.log4j.CloseableThreadContext.put;


public class HandlerS3 implements RequestHandler<S3Event, String>{
    private static final Logger logger = LoggerFactory.getLogger(HandlerS3.class);

    @Override
    public String handleRequest(S3Event event, Context context) {
        for(var record : event.getRecords()) {
            try (var loggingCtx = put("awsRegion", record.getAwsRegion())) {
                loggingCtx.put("eventName", record.getEventName());
```

```
                loggingCtx.put("bucket", record.getS3().getBucket().getName());
                loggingCtx.put("key", record.getS3().getObject().getKey());

                logger.info("Handling s3 event");
            }
        }

        return "Ok";
    }
}
```

Kode ini menghasilkan output log seperti berikut:

Example format log

```
{
    "timestamp": "2023-11-15T16:56:00.815Z",
    "level": "INFO",
    "message": "Handling s3 event",
    "logger": "example.HandlerS3",
    "AWSRequestId": "0bced576-3936-4e5a-9dcd-db9477b77f97",
    "awsRegion": "eu-south-1",
    "bucket": "java-logging-test-input-bucket",
    "eventName": "ObjectCreated:Put",
    "key": "test-folder/"
}
```

Konfigurasi build mengambil dependensi runtime pada appender Lambda dan adaptor SLF4J, dan dependensi implementasi pada Log4j2.

Example build.gradle – dependensi pencatatan

```
dependencies {
    ...
    'com.amazonaws:aws-lambda-java-log4j2:[1.6.0,)',
    'com.amazonaws:aws-lambda-java-events:[3.11.3,)',
    'org.apache.logging.log4j:log4j-layout-template-json:[2.17.1,)',
    'org.apache.logging.log4j:log4j-slf4j2-impl:[2.19.0,)',
    ...
}
```

Saat Anda menjalankan kode Anda secara lokal untuk pengujian, objek konteks dengan logger Lambda tidak tersedia, dan tidak ada ID permintaan untuk appender Lambda untuk digunakan. Misalnya konfigurasi uji, lihat aplikasi sampel di bagian berikutnya.

Menggunakan alat logging dan pustaka lainnya

Powertools for AWS Lambda (Java) adalah toolkit pengembang untuk mengimplementasikan praktik terbaik Tanpa Server dan meningkatkan kecepatan pengembang. Utilitas Logging menyediakan logger yang dioptimalkan Lambda yang mencakup informasi tambahan tentang konteks fungsi di semua fungsi Anda dengan output terstruktur sebagai JSON. Gunakan utilitas ini untuk melakukan hal berikut:

- Tangkap bidang kunci dari konteks Lambda, cold start, dan struktur logging output sebagai JSON
- Log peristiwa pemanggilan Lambda saat diinstruksikan (dinonaktifkan secara default)
- Cetak semua log hanya untuk persentase pemanggilan melalui pengambilan sampel log (dinonaktifkan secara default)
- Tambahkan kunci tambahan ke log terstruktur kapan saja
- Gunakan pemformat log kustom (Bring Your Own Formatter) untuk mengeluarkan log dalam struktur yang kompatibel dengan RFC Logging organisasi Anda

Menggunakan Powertools untuk AWS Lambda (Java) dan AWS SAM untuk logging terstruktur

Ikuti langkah-langkah di bawah ini untuk mengunduh, membangun, dan menyebarkan contoh aplikasi Hello World Java dengan Powertools terintegrasi untuk modul AWS Lambda (Java) menggunakan modul. AWS SAM Aplikasi ini mengimplementasikan backend API dasar dan menggunakan Powertools untuk memancarkan log, metrik, dan jejak. Ini terdiri dari titik akhir Amazon API Gateway dan Fungsi Peristiwa Lambda. Saat Anda mengirim permintaan GET ke titik akhir API Gateway, fungsi Lambda memanggil, mengirim log dan metrik menggunakan Format Metrik Tertanam CloudWatch ke, dan mengirimkan jejak ke. AWS X-Ray Fungsi mengembalikan `hello world` pesan.

Prasyarat

Untuk menyelesaikan langkah-langkah di bagian ini, Anda harus memiliki hal-hal berikut:

- Java 11
- AWS CLI versi 2

- AWS SAM CLI versi 1.75 atau yang lebih baru. Jika Anda memiliki versi CLI yang lebih lama, lihat Memutakhirkan AWS SAM CLI. AWS SAM

Menyebarkan sampel AWS SAM application

1. Inisialisasi aplikasi menggunakan template Hello World Java.

```
sam init --app-template hello-world-powertools-java --name sam-app --package-type
 Zip --runtime java11 --no-tracing
```

2. Bangun aplikasi.

```
cd sam-app && sam build
```

3. Terapkan aplikasi.

```
sam deploy --guided
```

4. Ikuti petunjuk di layar. Untuk menerima opsi default yang disediakan dalam pengalaman interaktif, tekan`Enter`.

> **Note**
>
> Karena HelloWorldFunction mungkin tidak memiliki otorisasi yang ditentukan, Apakah ini baik-baik saja? , pastikan untuk masuky.

5. Dapatkan URL aplikasi yang digunakan:

```
aws cloudformation describe-stacks --stack-name sam-app --query
 'Stacks[0].Outputs[?OutputKey==`HelloWorldApi`].OutputValue' --output text
```

6. Memanggil titik akhir API:

```
curl -X GET <URL_FROM_PREVIOUS_STEP>
```

Jika berhasil, Anda akan melihat tanggapan ini:

```
{"message":"hello world"}
```

7. Untuk mendapatkan log untuk fungsi tersebut, jalankan log sam. Untuk informasi selengkapnya, lihat Bekerja dengan log di Panduan AWS Serverless Application Model Pengembang.

```
sam logs --stack-name sam-app
```

Output log terlihat seperti ini:

```
2025/09/03/[$LATEST]851411a899b545eea2cffeba4cfbec81 2023-02-03T09:24:34.095000
 INIT_START Runtime Version: java:11.v15    Runtime Version ARN: arn:aws:lambda:eu-
central-1::runtime:0a25e3e7a1cc9ce404bc435eeb2ad358d8fa64338e618d0c224fe509403583ca
2025/09/03/[$LATEST]851411a899b545eea2cffeba4cfbec81 2023-02-03T09:24:34.114000
 Picked up JAVA_TOOL_OPTIONS: -XX:+TieredCompilation -XX:TieredStopAtLevel=1
2025/09/03/[$LATEST]851411a899b545eea2cffeba4cfbec81 2023-02-03T09:24:34.793000
 Transforming org/apache/logging/log4j/core/lookup/JndiLookup
 (lambdainternal.CustomerClassLoader@1a6c5a9e)
2025/09/03/[$LATEST]851411a899b545eea2cffeba4cfbec81 2023-02-03T09:24:35.252000
 START RequestId: 7fcf1548-d2d4-41cd-a9a8-6ae47c51f765 Version: $LATEST
2025/09/03/[$LATEST]851411a899b545eea2cffeba4cfbec81 2023-02-03T09:24:36.531000 {
  "_aws": {
    "Timestamp": 1675416276051,
    "CloudWatchMetrics": [
      {
        "Namespace": "sam-app-powerools-java",
        "Metrics": [
          {
            "Name": "ColdStart",
            "Unit": "Count"
          }
        ],
        "Dimensions": [
          [
            "Service",
            "FunctionName"
          ]
        ]
      }
    ]
  },
  "function_request_id": "7fcf1548-d2d4-41cd-a9a8-6ae47c51f765",
  "traceId":
 "Root=1-63dcd2d1-25f90b9d1c753a783547f4dd;Parent=e29684c1be352ce4;Sampled=1",
  "FunctionName": "sam-app-HelloWorldFunction-y9Iu1FLJJBGD",
```

```
  "functionVersion": "$LATEST",
  "ColdStart": 1.0,
  "Service": "service_undefined",
  "logStreamId": "2025/09/03/[$LATEST]851411a899b545eea2cffeba4cfbec81",
  "executionEnvironment": "AWS_Lambda_java11"
}
2025/09/03/[$LATEST]851411a899b545eea2cffeba4cfbec81 2023-02-03T09:24:36.974000 Feb
 03, 2023 9:24:36 AM com.amazonaws.xray.AWSXRayRecorder <init>
2025/09/03/[$LATEST]851411a899b545eea2cffeba4cfbec81 2023-02-03T09:24:36.993000 Feb
 03, 2023 9:24:36 AM com.amazonaws.xray.config.DaemonConfiguration <init>
2025/09/03/[$LATEST]851411a899b545eea2cffeba4cfbec81 2023-02-03T09:24:36.993000
 INFO: Environment variable AWS_XRAY_DAEMON_ADDRESS is set. Emitting to daemon on
 address XXXX.XXXX.XXXX.XXXX:2000.
2025/09/03/[$LATEST]851411a899b545eea2cffeba4cfbec81 2023-02-03T09:24:37.331000
 09:24:37.294 [main] INFO  helloworld.App - {"version":null,"resource":"/
hello","path":"/hello/","httpMethod":"GET","headers":{"Accept":"*/
*","CloudFront-Forwarded-Proto":"https","CloudFront-Is-Desktop-
Viewer":"true","CloudFront-Is-Mobile-Viewer":"false","CloudFront-Is-
SmartTV-Viewer":"false","CloudFront-Is-Tablet-Viewer":"false","CloudFront-
Viewer-ASN":"16509","CloudFront-Viewer-Country":"IE","Host":"XXXX.execute-
api.eu-central-1.amazonaws.com","User-Agent":"curl/7.86.0","Via":"2.0
 f0300a9921a99446a44423d996042050.cloudfront.net (CloudFront)","X-Amz-
Cf-Id":"t9W5ByT11HaY33NM8YioKECn_4eMpNsOMPfEVRczD7T1RdhbtiwV1Q==","X-
Amzn-Trace-Id":"Root=1-63dcd2d1-25f90b9d1c753a783547f4dd","X-Forwarded-
For":"XX.XXX.XXX.XX, XX.XXX.XXX.XX","X-Forwarded-Port":"443","X-
Forwarded-Proto":"https"},"multiValueHeaders":{"Accept":["*/
*"],"CloudFront-Forwarded-Proto":["https"],"CloudFront-Is-Desktop-Viewer":
["true"],"CloudFront-Is-Mobile-Viewer":["false"],"CloudFront-Is-SmartTV-
Viewer":["false"],"CloudFront-Is-Tablet-Viewer":["false"],"CloudFront-Viewer-
ASN":["16509"],"CloudFront-Viewer-Country":["IE"],"Host":["XXXX.execute-
api.eu-central-1.amazonaws.com"],"User-Agent":["curl/7.86.0"],"Via":["2.0
 f0300a9921a99446a44423d996042050.cloudfront.net (CloudFront)"],"X-Amz-
Cf-Id":["t9W5ByT11HaY33NM8YioKECn_4eMpNsOMPfEVRczD7T1RdhbtiwV1Q=="],"X-
Amzn-Trace-Id":["Root=1-63dcd2d1-25f90b9d1c753a783547f4dd"],"X-Forwarded-
For":["XXX, XXX"],"X-Forwarded-Port":["443"],"X-Forwarded-Proto":
["https"]},"queryStringParameters":null,"multiValueQueryStringParameters":null,"pathParamet
{"accountId":"XXX","stage":"Prod","resourceId":"at73a1","requestId":"ba09ecd2-
acf3-40f6-89af-fad32df67597","operationName":null,"identity":
{"cognitoIdentityPoolId":null,"accountId":null,"cognitoIdentityId":null,"caller":null,"apiK
hello","httpMethod":"GET","apiId":"XXX","path":"/Prod/
hello/","authorizer":null},"body":null,"isBase64Encoded":false}
2025/09/03/[$LATEST]851411a899b545eea2cffeba4cfbec81 2023-02-03T09:24:37.351000
 09:24:37.351 [main] INFO  helloworld.App - Retrieving https://
checkip.amazonaws.com
```

```
2025/09/03/[$LATEST]851411a899b545eea2cffeba4cfbec81 2023-02-03T09:24:39.313000 {
  "function_request_id": "7fcf1548-d2d4-41cd-a9a8-6ae47c51f765",
  "traceId":
 "Root=1-63dcd2d1-25f90b9d1c753a783547f4dd;Parent=e29684c1be352ce4;Sampled=1",
  "xray_trace_id": "1-63dcd2d1-25f90b9d1c753a783547f4dd",
  "functionVersion": "$LATEST",
  "Service": "service_undefined",
  "logStreamId": "2025/09/03/[$LATEST]851411a899b545eea2cffeba4cfbec81",
  "executionEnvironment": "AWS_Lambda_java11"
}
2025/09/03/[$LATEST]851411a899b545eea2cffeba4cfbec81 2023-02-03T09:24:39.371000 END
 RequestId: 7fcf1548-d2d4-41cd-a9a8-6ae47c51f765
2025/09/03/[$LATEST]851411a899b545eea2cffeba4cfbec81 2023-02-03T09:24:39.371000
 REPORT RequestId: 7fcf1548-d2d4-41cd-a9a8-6ae47c51f765    Duration: 4118.98 ms
 Billed Duration: 5275 ms    Memory Size: 512 MB    Max Memory Used: 152 MB    Init
 Duration: 1155.47 ms
XRAY TraceId: 1-63dcd2d1-25f90b9d1c753a783547f4dd    SegmentId: 3a028fee19b895cb
 Sampled: true
```

8. Ini adalah titik akhir API publik yang dapat diakses melalui internet. Kami menyarankan Anda menghapus titik akhir setelah pengujian.

```
sam delete
```

Mengelola retensi log

Grup log tidak terhapus secara otomatis ketika Anda menghapus suatu fungsi. Untuk menghindari penyimpanan log tanpa batas waktu, hapus grup log, atau konfigurasikan periode retensi setelah itu secara CloudWatch otomatis menghapus log. Untuk mengatur penyimpanan log, tambahkan yang berikut ini ke AWS SAM templat Anda:

```
Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function
    Properties:
      # Omitting other properties

  LogGroup:
    Type: AWS::Logs::LogGroup
    Properties:
      LogGroupName: !Sub "/aws/lambda/${HelloWorldFunction}"
```

```
      RetentionInDays: 7
```

Melihat log di konsol Lambda

Anda dapat menggunakan konsol Lambda untuk melihat keluaran log setelah Anda menjalankan Fungsi Peristiwa Lambda.

Jika kode Anda dapat diuji dari editor Kode tertanam, Anda menemukan log dalam hasil eksekusi. Saat Anda menggunakan fitur pengujian konsol untuk menjalankan fungsi, Anda menemukan Keluaran Log di bagian Detail.

Melihat log di CloudWatch konsol

Anda dapat menggunakan CloudWatch konsol Amazon untuk melihat log untuk semua pemanggilan Fungsi Peristiwa Lambda.

Untuk melihat log di CloudWatch konsol

1. Buka halaman Grup log di CloudWatch konsol.
2. Pilih grup log untuk fungsi Anda (/aws/lambda/***your-function-name***).
3. Pilih pengaliran log.

Setiap aliran log sesuai dengan instans fungsi Anda. Aliran log muncul saat Anda memperbarui fungsi Lambda, dan saat instance tambahan dibuat untuk menangani pemanggilan bersamaan. Untuk menemukan log untuk pemanggilan tertentu, kami sarankan untuk menginstrumentasi fungsi Anda dengan. AWS X-Ray X-Ray mencatat detail tentang permintaan dan aliran log dalam jejak.

Melihat log menggunakan AWS Command Line Interface (AWS CLI)

AWS CLI Ini adalah alat sumber terbuka yang dapat Anda gunakan untuk berinteraksi dengan AWS layanan menggunakan perintah di shell baris perintah Anda. Untuk menyelesaikan langkah-langkah di bagian ini, Anda harus memiliki AWS CLI versi 2.

Anda dapat menggunakan AWS CLI untuk mengambil log untuk invokasi menggunakan opsi perintah `--log-type`. Respons berisi bidang `LogResult` yang memuat hingga 4 KB log berkode base64 dari invokasi.

Example mengambil ID log

Contoh berikut menunjukkan cara mengambil ID log dari `LogResult` untuk fungsi bernama `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

Anda akan melihat output berikut:

```
{
    "StatusCode": 200,
    "LogResult":
 "U1RBUlQgUmVxdWVzdElkOiA4N2QwNDRiOC1mMTU0LTExZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "ExecutedVersion": "$LATEST"
}
```

Example mendekode log

Pada prompt perintah yang sama, gunakan utilitas `base64` untuk mendekodekan log. Contoh berikut menunjukkan cara mengambil log berkode base64 untuk `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail \
--query 'LogResult' --output text --cli-binary-format raw-in-base64-out | base64 --
decode
```

cli-binary-formatOpsi ini diperlukan jika Anda menggunakan AWS CLI versi 2. Untuk menjadikan ini pengaturan default, jalankan`aws configure set cli-binary-format raw-in-base64-out`. Untuk informasi selengkapnya, lihat opsi baris perintah global yang AWS CLI didukung di Panduan AWS Command Line Interface Pengguna untuk Versi 2.

Anda akan melihat output berikut:

```
START RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Version: $LATEST
"AWS_SESSION_TOKEN": "AgoJb3JpZ2luX2VjELj...", "_X_AMZN_TRACE_ID": "Root=1-5d02e5ca-
f5792818b6fe8368e5b51d50;Parent=191db58857df8395;Sampled=0"",ask/lib:/opt/lib",
END RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8
REPORT RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8  Duration: 79.67 ms      Billed
 Duration: 80 ms         Memory Size: 128 MB     Max Memory Used: 73 MB
```

Utilitas `base64` tersedia di Linux, macOS, dan Ubuntu pada Windows. Pengguna macOS mungkin harus menggunakan `base64 -D`.

Example Skrip get-logs.sh

Pada prompt perintah yang sama, gunakan script berikut untuk mengunduh lima peristiwa log terakhir. Skrip menggunakan `sed` untuk menghapus kutipan dari file output, dan akan tidur selama 15

detik untuk memberikan waktu agar log tersedia. Output mencakup respons dari Lambda dan output dari perintah `get-log-events`.

Salin konten dari contoh kode berikut dan simpan dalam direktori proyek Lambda Anda sebagai `get-logs.sh`.

cli-binary-formatOpsi ini diperlukan jika Anda menggunakan AWS CLI versi 2. Untuk menjadikan ini pengaturan default, jalankan`aws configure set cli-binary-format raw-in-base64-out`. Untuk informasi selengkapnya, lihat opsi baris perintah global yang AWS CLI didukung di Panduan AWS Command Line Interface Pengguna untuk Versi 2.

```
#!/bin/bash
aws lambda invoke --function-name my-function --cli-binary-format raw-in-base64-out --
payload '{"key": "value"}' out
sed -i'' -e 's/"//g' out
sleep 15
aws logs get-log-events --log-group-name /aws/lambda/my-function --log-stream-
name stream1 --limit 5
```

Example macOS dan Linux (khusus)

Dalam prompt perintah yang sama, pengguna macOS dan Linux mungkin perlu menjalankan perintah berikut untuk memastikan skrip dapat dieksekusi.

```
chmod -R 755 get-logs.sh
```

Example mengambil lima log acara terakhir

Pada prompt perintah yang sama, gunakan skrip berikut untuk mendapatkan lima log acara terakhir.

```
./get-logs.sh
```

Anda akan melihat output berikut:

```
{
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}
{
```

```
    "events": [
        {
            "timestamp": 1559763003171,
            "message": "START RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf Version:
 $LATEST\n",
            "ingestionTime": 1559763003309
        },
        {
            "timestamp": 1559763003173,
            "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tENVIRONMENT VARIABLES\r{\r  \"AWS_LAMBDA_FUNCTION_VERSION\": \"$LATEST\",
\r ...",
            "ingestionTime": 1559763018353
        },
        {
            "timestamp": 1559763003173,
            "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tEVENT\r{\r  \"key\": \"value\"\r}\n",
            "ingestionTime": 1559763018353
        },
        {
            "timestamp": 1559763003218,
            "message": "END RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf\n",
            "ingestionTime": 1559763018353
        },
        {
            "timestamp": 1559763003218,
            "message": "REPORT RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tDuration: 26.73 ms\tBilled Duration: 27 ms \tMemory Size: 128 MB\tMax Memory Used: 75
 MB\t\n",
            "ingestionTime": 1559763018353
        }
    ],
    "nextForwardToken": "f/34783877304859518393868359594929986069206639495374241795",
    "nextBackwardToken": "b/34783877303811383369537420289090800615709599058929582080"
}
```


Menghapus log

Grup log tidak terhapus secara otomatis ketika Anda menghapus suatu fungsi. Untuk menghindari penyimpanan log secara tidak terbatas, hapus kelompok log, atau lakukan konfigurasi periode penyimpanan, yang setelahnya log akan dihapus secara otomatis.

Kode pencatatan sampel

GitHub Repositori untuk panduan ini mencakup contoh aplikasi yang menunjukkan penggunaan berbagai konfigurasi logging. Setiap contoh aplikasi menyertakan skrip untuk penyebaran dan pembersihan yang mudah, AWS SAM templat, dan sumber daya pendukung.

Sampel aplikasi Lambda di Java

- example-java - Fungsi Acara Java yang menunjukkan bagaimana Anda dapat menggunakan Lambda untuk memproses pesanan. Fungsi ini menggambarkan cara mendefinisikan dan deserialisasi objek peristiwa input kustom, menggunakan AWS SDK, dan logging keluaran.
- java-basic - Kumpulan fungsi Java minimal dengan pengujian unit dan konfigurasi logging variabel.
- java-events - Kumpulan fungsi Java yang berisi kode kerangka untuk cara menangani peristiwa dari berbagai layanan seperti Amazon API Gateway, Amazon SQS, dan Amazon Kinesis. Fungsi-fungsi ini menggunakan versi terbaru dari pustaka aws-lambda-java-events (3.0.0 dan yang lebih baru). Contoh-contoh ini tidak memerlukan AWS SDK sebagai dependensi.
- s3-java – Fungsi Java yang memproses kejadian pemberitahuan dari Amazon S3 dan menggunakan Java Class Library (JCL) untuk membuat thumbnail dari file gambar yang diunggah.
- layer-java — Fungsi Java yang menggambarkan cara menggunakan lapisan Lambda untuk mengemas dependensi yang terpisah dari kode fungsi inti Anda.

Aplikasi sampel `java-basic` menunjukkan konfigurasi pencatatan minimal yang mendukung uji pencatatan. Kode handler menggunakan logger `LambdaLogger` yang disediakan oleh objek konteks. Untuk pengujian, aplikasi menggunakan `TestLogger` kelas khusus yang mengimplementasikan `LambdaLogger` antarmuka dengan logger Log4j2. Ini menggunakan SLF4J sebagai fasad untuk kompatibilitas dengan SDK. AWS Pustaka log masuk dikecualikan dari output pembuatan untuk menjaga paket deployment tetap kecil.

Menginstrumentasi kode Java di AWS Lambda

Lambda terintegrasi dengan AWS X-Ray untuk membantu Anda melacak, men-debug, dan mengoptimalkan aplikasi Lambda. Anda dapat menggunakan X-Ray untuk melacak permintaan saat melintasi sumber daya dalam aplikasi Anda, yang mungkin termasuk fungsi Lambda dan layanan lainnya. AWS

Untuk mengirim data penelusuran X-Ray, Anda dapat menggunakan salah satu dari dua pustaka SDK:

- AWS Distro for OpenTelemetry (ADOT) — Distribusi SDK (OTel) yang aman, siap produksi, dan AWS didukung. OpenTelemetry
- AWS X-Ray SDK for Java— SDK untuk menghasilkan dan mengirim data jejak ke X-Ray.
- Powertools for AWS Lambda (Java) - Toolkit pengembang untuk menerapkan praktik terbaik Tanpa Server dan meningkatkan kecepatan pengembang.

Setiap SDK menawarkan cara untuk mengirim data telemetri Anda ke layanan. X-Ray Anda kemudian dapat menggunakan X-Ray untuk melihat, memfilter, dan mendapatkan wawasan tentang metrik kinerja aplikasi Anda untuk mengidentifikasi masalah dan peluang untuk pengoptimalan.

> **Important**
>
> The X-Ray dan Powertools untuk AWS Lambda SDK adalah bagian dari solusi instrumentasi terintegrasi ketat yang ditawarkan oleh. AWS Lapisan Lambda ADOT adalah bagian dari standar industri untuk melacak instrumentasi yang mengumpulkan lebih banyak data secara umum, tetapi mungkin tidak cocok untuk semua kasus penggunaan. Anda dapat menerapkan penelusuran ujung ke ujung dalam X-Ray menggunakan salah satu solusi. Untuk mempelajari lebih lanjut tentang memilih di antara keduanya, lihat Memilih antara AWS Distro untuk Telemetri Terbuka dan SDK. X-Ray

Bagian-bagian

- Menggunakan Powertools untuk AWS Lambda (Java) dan AWS SAM untuk menelusuri
- Menggunakan Powertools untuk AWS Lambda (Java) dan AWS CDK untuk menelusuri
- Menggunakan ADOT untuk instrumen fungsi Java Anda
- Menggunakan X-Ray SDK untuk instrumen fungsi Java Anda

- Mengaktifkan penelusuran dengan konsol Lambda
- Mengaktifkan penelusuran dengan Lambda API
- Mengaktifkan penelusuran dengan CloudFormation
- Menafsirkan jejak X-Ray
- Menyimpan dependensi runtime dalam lapisan (SDK) X-Ray
- X-Ray penelusuran dalam aplikasi sampel (X-Ray SDK)

Menggunakan Powertools untuk AWS Lambda (Java) dan AWS SAM untuk menelusuri

Ikuti langkah-langkah di bawah ini untuk mengunduh, membangun, dan menyebarkan contoh aplikasi Hello World Java dengan Powertools terintegrasi untuk modul AWS Lambda (Java) menggunakan modul. AWS SAM Aplikasi ini mengimplementasikan backend API dasar dan menggunakan Powertools untuk memancarkan log, metrik, dan jejak. Ini terdiri dari titik akhir Amazon API Gateway dan Fungsi Peristiwa Lambda. Saat Anda mengirim permintaan GET ke titik akhir API Gateway, fungsi Lambda memanggil, mengirim log dan metrik menggunakan Format Metrik Tertanam CloudWatch ke, dan mengirimkan jejak ke. AWS X-Ray Fungsi mengembalikan `hello world` pesan.

Prasyarat

Untuk menyelesaikan langkah-langkah di bagian ini, Anda harus memiliki hal-hal berikut:

- Java 11 atau yang lebih baru
- AWS CLI versi 2
- AWS SAM CLI versi 1.75 atau yang lebih baru. Jika Anda memiliki versi CLI yang lebih lama, lihat Memutakhirkan AWS SAM CLI. AWS SAM

Menyebarkan sampel AWS SAM application

1. Inisialisasi aplikasi menggunakan template Hello World Java.

```
sam init --app-template hello-world-powertools-java --name sam-app --package-type
 Zip --runtime java11 --no-tracing
```

2. Bangun aplikasi.

```
cd sam-app && sam build
```

3. Terapkan aplikasi.

```
sam deploy --guided
```

4. Ikuti petunjuk di layar. Untuk menerima opsi default yang disediakan dalam pengalaman interaktif, tekan`Enter`.

> **Note**
>
> Karena HelloWorldFunction mungkin tidak memiliki otorisasi yang ditentukan, Apakah ini baik-baik saja? , pastikan untuk masuky.

5. Dapatkan URL aplikasi yang digunakan:

```
aws cloudformation describe-stacks --stack-name sam-app --query
 'Stacks[0].Outputs[?OutputKey==`HelloWorldApi`].OutputValue' --output text
```

6. Memanggil titik akhir API:

```
curl -X GET <URL_FROM_PREVIOUS_STEP>
```

Jika berhasil, Anda akan melihat tanggapan ini:

```
{"message":"hello world"}
```

7. Untuk mendapatkan jejak untuk fungsi tersebut, jalankan jejak sam.

```
sam traces
```

Output jejak terlihat seperti ini:

```
New XRay Service Graph
  Start time: 2025-02-03 14:31:48+01:00
  End time: 2025-02-03 14:31:48+01:00
  Reference Id: 0 - (Root) AWS::Lambda - sam-app-HelloWorldFunction-y9Iu1FLJJBGD -
 Edges: []
   Summary_statistics:
     - total requests: 1
     - ok count(2XX): 1
     - error count(4XX): 0
```

```
    - fault count(5XX): 0
    - total response time: 5.587
  Reference Id: 1 - client - sam-app-HelloWorldFunction-y9Iu1FLJJBGD - Edges: [0]
   Summary_statistics:
    - total requests: 0
    - ok count(2XX): 0
    - error count(4XX): 0
    - fault count(5XX): 0
    - total response time: 0

XRay Event [revision 3] at (2025-02-03T14:31:48.500000) with id
 (1-63dd0cc4-3c869dec72a586875da39777) and duration (5.603s)
 - 5.587s - sam-app-HelloWorldFunction-y9Iu1FLJJBGD [HTTP: 200]
 - 4.053s - sam-app-HelloWorldFunction-y9Iu1FLJJBGD
   - 1.181s - Initialization
   - 4.037s - Invocation
     - 1.981s - ## handleRequest
       - 1.840s - ## getPageContents
   - 0.000s - Overhead
```

8. Ini adalah titik akhir API publik yang dapat diakses melalui internet. Kami menyarankan Anda menghapus titik akhir setelah pengujian.

```
sam delete
```

Menggunakan Powertools untuk AWS Lambda (Java) dan AWS CDK untuk menelusuri

Ikuti langkah-langkah di bawah ini untuk mengunduh, membangun, dan menyebarkan contoh aplikasi Hello World Java dengan Powertools terintegrasi untuk modul AWS Lambda (Java) menggunakan modul. AWS CDK Aplikasi ini mengimplementasikan backend API dasar dan menggunakan Powertools untuk memancarkan log, metrik, dan jejak. Ini terdiri dari titik akhir Amazon API Gateway dan Fungsi Peristiwa Lambda. Saat Anda mengirim permintaan GET ke titik akhir API Gateway, fungsi Lambda memanggil, mengirim log dan metrik menggunakan Format Metrik Tertanam CloudWatch ke, dan mengirimkan jejak ke. AWS X-Ray Fungsi mengembalikan pesan hello world.

Prasyarat

Untuk menyelesaikan langkah-langkah di bagian ini, Anda harus memiliki hal-hal berikut:

- Java 11 atau yang lebih baru
- AWS CLI versi 2

- AWS CDK versi 2
- AWS SAM CLI versi 1.75 atau yang lebih baru. Jika Anda memiliki versi CLI yang lebih lama, lihat Memutakhirkan AWS SAM CLI. AWS SAM

Menyebarkan sampel AWS CDK application

1. Buat direktori proyek untuk aplikasi baru Anda.

```
mkdir hello-world
cd hello-world
```

2. Inisialisasi aplikasi.

```
cdk init app --language java
```

3. Buat proyek maven dengan perintah berikut:

```
mkdir app
cd app
mvn archetype:generate -DgroupId=helloworld -DartifactId=Function -
DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
```

4. Buka `pom.xml` di `hello-world\app\Function` direktori dan ganti kode yang ada dengan kode berikut yang mencakup dependensi dan plugin maven untuk Powertools.

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/
maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>helloworld</groupId>
  <artifactId>Function</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>Function</name>
  <url>http://maven.apache.org</url>
<properties>
    <maven.compiler.source>11</maven.compiler.source>
    <maven.compiler.target>11</maven.compiler.target>
    <log4j.version>2.17.2</log4j.version>
</properties>
```

```
    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>3.8.1</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>software.amazon.lambda</groupId>
            <artifactId>powertools-tracing</artifactId>
            <version>1.3.0</version>
        </dependency>
        <dependency>
            <groupId>software.amazon.lambda</groupId>
            <artifactId>powertools-metrics</artifactId>
            <version>1.3.0</version>
        </dependency>
        <dependency>
            <groupId>software.amazon.lambda</groupId>
            <artifactId>powertools-logging</artifactId>
            <version>1.3.0</version>
        </dependency>
        <dependency>
            <groupId>com.amazonaws</groupId>
            <artifactId>aws-lambda-java-core</artifactId>
            <version>1.2.2</version>
        </dependency>
        <dependency>
            <groupId>com.amazonaws</groupId>
            <artifactId>aws-lambda-java-events</artifactId>
            <version>3.11.1</version>
        </dependency>
  </dependencies>
<build>
    <plugins>
        <plugin>
            <groupId>org.codehaus.mojo</groupId>
            <artifactId>aspectj-maven-plugin</artifactId>
            <version>1.14.0</version>
            <configuration>
                <source>${maven.compiler.source}</source>
                <target>${maven.compiler.target}</target>
                <complianceLevel>${maven.compiler.target}</complianceLevel>
                <aspectLibraries>
```

```
                    <aspectLibrary>
                        <groupId>software.amazon.lambda</groupId>
                        <artifactId>powertools-tracing</artifactId>
                    </aspectLibrary>
                    <aspectLibrary>
                        <groupId>software.amazon.lambda</groupId>
                        <artifactId>powertools-metrics</artifactId>
                    </aspectLibrary>
                    <aspectLibrary>
                        <groupId>software.amazon.lambda</groupId>
                        <artifactId>powertools-logging</artifactId>
                    </aspectLibrary>
                </aspectLibraries>
            </configuration>
            <executions>
                <execution>
                    <goals>
                        <goal>compile</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
        <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-shade-plugin</artifactId>
                <version>3.4.1</version>
                <executions>
                    <execution>
                        <phase>package</phase>
                        <goals>
                            <goal>shade</goal>
                        </goals>
                        <configuration>
                            <transformers>
                                <transformer

 implementation="com.github.edwgiz.maven_shade_plugin.log4j2_cache_transformer.PluginsCache
                                </transformer>
                            </transformers>
                            <createDependencyReducedPom>false</
createDependencyReducedPom>
                            <finalName>function</finalName>

                        </configuration>
```

```
                    </execution>
                </executions>
                <dependencies>
                    <dependency>
                        <groupId>com.github.edwgiz</groupId>
                        <artifactId>maven-shade-plugin.log4j2-cachefile-
transformer</artifactId>
                        <version>2.15</version>
                    </dependency>
                </dependencies>
        </plugin>
    </plugins>
</build>
</project>
```

5. Buat `hello-world\app\src\main\resource` direktori dan buat `log4j.xml` untuk konfigurasi log.

```
mkdir -p src/main/resource
cd src/main/resource
touch log4j.xml
```

6. Buka `log4j.xml` dan tambahkan kode berikut.

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration>
    <Appenders>
        <Console name="JsonAppender" target="SYSTEM_OUT">
            <JsonTemplateLayout
 eventTemplateUri="classpath:LambdaJsonLayout.json" />
        </Console>
    </Appenders>
    <Loggers>
        <Logger name="JsonLogger" level="INFO" additivity="false">
            <AppenderRef ref="JsonAppender"/>
        </Logger>
        <Root level="info">
            <AppenderRef ref="JsonAppender"/>
        </Root>
    </Loggers>
</Configuration>
```

7. Buka `App.java` dari `hello-world\app\Function\src\main\java\helloworld` direktori dan ganti kode yang ada dengan kode berikut. Ini adalah kode untuk fungsi Lambda.

```
package helloworld;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.URL;
import java.util.HashMap;
import java.util.Map;
import java.util.stream.Collectors;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.APIGatewayProxyRequestEvent;
import com.amazonaws.services.lambda.runtime.events.APIGatewayProxyResponseEvent;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import software.amazon.lambda.powertools.logging.Logging;
import software.amazon.lambda.powertools.metrics.Metrics;
import software.amazon.lambda.powertools.tracing.CaptureMode;
import software.amazon.lambda.powertools.tracing.Tracing;

import static software.amazon.lambda.powertools.tracing.CaptureMode.*;

/**
 * Handler for requests to Lambda function.
 */
public class App implements RequestHandler<APIGatewayProxyRequestEvent,
 APIGatewayProxyResponseEvent> {
    Logger log = LogManager.getLogger(App.class);


    @Logging(logEvent = true)
    @Tracing(captureMode = DISABLED)
    @Metrics(captureColdStart = true)
    public APIGatewayProxyResponseEvent handleRequest(final
 APIGatewayProxyRequestEvent input, final Context context) {
        Map<String, String> headers = new HashMap<>();
        headers.put("Content-Type", "application/json");
        headers.put("X-Custom-Header", "application/json");
```

```
        APIGatewayProxyResponseEvent response = new APIGatewayProxyResponseEvent()
                .withHeaders(headers);
        try {
            final String pageContents = this.getPageContents("https://
checkip.amazonaws.com");
            String output = String.format("{ \"message\": \"hello world\",
 \"location\": \"%s\" }", pageContents);

            return response
                    .withStatusCode(200)
                    .withBody(output);
        } catch (IOException e) {
            return response
                    .withBody("{}")
                    .withStatusCode(500);
        }
    }
    @Tracing(namespace = "getPageContents")
    private String getPageContents(String address) throws IOException {
        log.info("Retrieving {}", address);
        URL url = new URL(address);
        try (BufferedReader br = new BufferedReader(new
 InputStreamReader(url.openStream()))) {
            return br.lines().collect(Collectors.joining(System.lineSeparator()));
        }
    }
}
```

8. Buka `HelloWorldStack.java` dari `hello-world\src\main\java\com\myorg` direktori dan ganti kode yang ada dengan kode berikut. Kode ini menggunakan Lambda Constructor dan ApiGatewayv2 Constructor untuk membuat REST API dan Lambda Event Function.

```
package com.myorg;

import software.amazon.awscdk.*;
import software.amazon.awscdk.services.apigatewayv2.alpha.*;
import
 software.amazon.awscdk.services.apigatewayv2.integrations.alpha.HttpLambdaIntegration;
import
 software.amazon.awscdk.services.apigatewayv2.integrations.alpha.HttpLambdaIntegrationProps
import software.amazon.awscdk.services.lambda.Code;
import software.amazon.awscdk.services.lambda.Function;
import software.amazon.awscdk.services.lambda.FunctionProps;
```

```
import software.amazon.awscdk.services.lambda.Runtime;
import software.amazon.awscdk.services.lambda.Tracing;
import software.amazon.awscdk.services.logs.RetentionDays;
import software.amazon.awscdk.services.s3.assets.AssetOptions;
import software.constructs.Construct;

import java.util.Arrays;
import java.util.List;

import static java.util.Collections.singletonList;
import static software.amazon.awscdk.BundlingOutput.ARCHIVED;

public class HelloWorldStack extends Stack {
    public HelloWorldStack(final Construct scope, final String id) {
        this(scope, id, null);
    }

    public HelloWorldStack(final Construct scope, final String id, final StackProps
 props) {
        super(scope, id, props);

        List<String> functionPackagingInstructions = Arrays.asList(
                "/bin/sh",
                "-c",
                "cd Function " +
                        "&& mvn clean install " +
                        "&& cp /asset-input/Function/target/function.jar /asset-
output/"
        );
        BundlingOptions.Builder builderOptions = BundlingOptions.builder()
                .command(functionPackagingInstructions)
                .image(Runtime.JAVA_11.getBundlingImage())
                .volumes(singletonList(
                        // Mount local .m2 repo to avoid download all the
 dependencies again inside the container
                        DockerVolume.builder()
                                .hostPath(System.getProperty("user.home") +
 "/.m2/")
                                .containerPath("/root/.m2/")
                                .build()
                ))
                .user("root")
                .outputType(ARCHIVED);
```

```
        Function function = new Function(this, "Function", FunctionProps.builder()
                .runtime(Runtime.JAVA_11)
                .code(Code.fromAsset("app", AssetOptions.builder()
                        .bundling(builderOptions
                                .command(functionPackagingInstructions)
                                .build())
                        .build()))
                .handler("helloworld.App::handleRequest")
                .memorySize(1024)
                .tracing(Tracing.ACTIVE)
                .timeout(Duration.seconds(10))
                .logRetention(RetentionDays.ONE_WEEK)
                .build());

        HttpApi httpApi = new HttpApi(this, "sample-api", HttpApiProps.builder()
                .apiName("sample-api")
                .build());

        httpApi.addRoutes(AddRoutesOptions.builder()
                .path("/")
                .methods(singletonList(HttpMethod.GET))
                .integration(new HttpLambdaIntegration("function", function,
 HttpLambdaIntegrationProps.builder()
                        .payloadFormatVersion(PayloadFormatVersion.VERSION_2_0)
                        .build()))
                .build());

        new CfnOutput(this, "HttpApi", CfnOutputProps.builder()
                .description("Url for Http Api")
                .value(httpApi.getApiEndpoint())
                .build());
    }
}
```

9. Buka pom.xml dari hello-world direktori dan ganti kode yang ada dengan kode berikut.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd"
         xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.myorg</groupId>
```

```
    <artifactId>hello-world</artifactId>
    <version>0.1</version>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <cdk.version>2.70.0</cdk.version>
        <constructs.version>[10.0.0,11.0.0)</constructs.version>
        <junit.version>5.7.1</junit.version>
    </properties>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.8.1</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>

            <plugin>
                <groupId>org.codehaus.mojo</groupId>
                <artifactId>exec-maven-plugin</artifactId>
                <version>3.0.0</version>
                <configuration>
                    <mainClass>com.myorg.HelloWorldApp</mainClass>
                </configuration>
            </plugin>
        </plugins>
    </build>

    <dependencies>
        <!-- AWS Cloud Development Kit -->
        <dependency>
            <groupId>software.amazon.awscdk</groupId>
            <artifactId>aws-cdk-lib</artifactId>
            <version>${cdk.version}</version>
        </dependency>
        <dependency>
            <groupId>software.constructs</groupId>
            <artifactId>constructs</artifactId>
            <version>${constructs.version}</version>
```

```
        </dependency>
        <dependency>
          <groupId>org.junit.jupiter</groupId>
          <artifactId>junit-jupiter</artifactId>
          <version>${junit.version}</version>
          <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>software.amazon.awscdk</groupId>
            <artifactId>apigatewayv2-alpha</artifactId>
            <version>${cdk.version}-alpha.0</version>
        </dependency>
        <dependency>
            <groupId>software.amazon.awscdk</groupId>
            <artifactId>apigatewayv2-integrations-alpha</artifactId>
            <version>${cdk.version}-alpha.0</version>
        </dependency>
    </dependencies>
</project>
```

10. Pastikan Anda berada di `hello-world` direktori dan menyebarkan aplikasi Anda.

```
cdk deploy
```

11. Dapatkan URL aplikasi yang digunakan:

```
aws cloudformation describe-stacks --stack-name HelloWorldStack --query
 'Stacks[0].Outputs[?OutputKey==`HttpApi`].OutputValue' --output text
```

12. Memanggil titik akhir API:

```
curl -X GET <URL_FROM_PREVIOUS_STEP>
```

    Jika berhasil, Anda akan melihat tanggapan ini:

```
{"message":"hello world"}
```

13. Untuk mendapatkan jejak untuk fungsi tersebut, jalankan jejak sam.

```
sam traces
```

    Output jejak terlihat seperti ini:

```
New XRay Service Graph
  Start time: 2025-02-03 14:59:50+00:00
  End time: 2025-02-03 14:59:50+00:00
  Reference Id: 0 - (Root) AWS::Lambda - sam-app-HelloWorldFunction-YBg8yfYtOc9j -
 Edges: [1]
   Summary_statistics:
     - total requests: 1
     - ok count(2XX): 1
     - error count(4XX): 0
     - fault count(5XX): 0
     - total response time: 0.924
  Reference Id: 1 - AWS::Lambda::Function - sam-app-HelloWorldFunction-YBg8yfYtOc9j
 - Edges: []
   Summary_statistics:
     - total requests: 1
     - ok count(2XX): 1
     - error count(4XX): 0
     - fault count(5XX): 0
     - total response time: 0.016
  Reference Id: 2 - client - sam-app-HelloWorldFunction-YBg8yfYtOc9j - Edges: [0]
   Summary_statistics:
     - total requests: 0
     - ok count(2XX): 0
     - error count(4XX): 0
     - fault count(5XX): 0
     - total response time: 0

XRay Event [revision 1] at (2025-02-03T14:59:50.204000) with id
 (1-63dd2166-434a12c22e1307ff2114f299) and duration (0.924s)
 - 0.924s - sam-app-HelloWorldFunction-YBg8yfYtOc9j [HTTP: 200]
 - 0.016s - sam-app-HelloWorldFunction-YBg8yfYtOc9j
   - 0.739s - Initialization
   - 0.016s - Invocation
     - 0.013s - ## lambda_handler
       - 0.000s - ## app.hello
   - 0.000s - Overhead
```

14. Ini adalah titik akhir API publik yang dapat diakses melalui internet. Kami menyarankan Anda menghapus titik akhir setelah pengujian.

```
cdk destroy
```

Menggunakan ADOT untuk instrumen fungsi Java Anda

ADOT menyediakan lapisan Lambda yang dikelola sepenuhnya yang mengemas semua yang Anda butuhkan untuk mengumpulkan data telemetri menggunakan OTel SDK. Dengan menggunakan lapisan ini, Anda dapat menginstruksikan fungsi Lambda Anda tanpa harus memodifikasi kode fungsi apa pun. Anda juga dapat mengonfigurasi lapisan Anda untuk melakukan inisialisasi khusus OTel. Untuk informasi selengkapnya, lihat Konfigurasi khusus untuk Kolektor ADOT di Lambda dalam dokumentasi ADOT.

Untuk runtime Java, Anda dapat memilih antara dua lapisan untuk dikonsumsi:

- AWS lapisan Lambda terkelola untuk ADOT Java (Auto-instrumentation Agen) - Lapisan ini secara otomatis mengubah kode fungsi Anda saat startup untuk mengumpulkan data penelusuran. Untuk petunjuk rinci tentang cara menggunakan layer ini bersama dengan agen ADOT Java, lihat AWS Distro untuk Lambda OpenTelemetry Support for Java (Auto-instrumentation Agent) dalam dokumentasi ADOT.
- AWS Lapisan Lambda terkelola untuk ADOT Java - Lapisan ini juga menyediakan instrumentasi bawaan untuk fungsi Lambda, tetapi memerlukan beberapa perubahan kode manual untuk menginisialisasi SDK OTel. Untuk petunjuk rinci tentang cara menggunakan layer ini, lihat AWS Distro untuk OpenTelemetry Lambda Support for Java di dokumentasi ADOT.

Untuk petunjuk tentang mengonversi peristiwa OpenTelemetry API Telemetri Lambda ke rentang, lihat. the section called “Mengonversi acara ke OTel Spans”

Menggunakan X-Ray SDK untuk instrumen fungsi Java Anda

Untuk merekam data tentang panggilan yang dilakukan fungsi Anda ke sumber daya dan layanan lain dalam aplikasi Anda, Anda dapat menambahkan X-Ray SDK for Java ke konfigurasi build Anda. Contoh berikut menunjukkan konfigurasi build Gradle yang menyertakan pustaka yang mengaktifkan instrumentasi otomatis klien. AWS SDK for Java 2.x

Example build.gradle — Menelusuri dependensi

```
dependencies {
    implementation platform('software.amazon.awssdk:bom:2.16.1')
    implementation platform('com.amazonaws:aws-xray-recorder-sdk-bom:2.11.0')
    ...
    implementation 'com.amazonaws:aws-xray-recorder-sdk-core'
    implementation 'com.amazonaws:aws-xray-recorder-sdk-aws-sdk'
```

```
    implementation 'com.amazonaws:aws-xray-recorder-sdk-aws-sdk-v2-instrumentor'
    ...
}
```

Setelah Anda menambahkan dependensi yang benar dan membuat perubahan kode yang diperlukan, aktifkan penelusuran dalam konfigurasi fungsi Anda melalui konsol Lambda atau API.

Mengaktifkan penelusuran dengan konsol Lambda

Untuk mengaktifkan penelusuran aktif di Fungsi Peristiwa Lambda Anda dengan konsol, ikuti langkah-langkah berikut:

Untuk mengaktifkan penelusuran aktif

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih Fungsi Acara.
3. Pilih Konfigurasi dan kemudian pilih Alat Pemantauan dan operasi.
4. Di bawah Alat pemantauan tambahan, pilih Edit.
5. Di bawah Sinyal CloudWatch Aplikasi dan AWS X-Ray, pilih Aktifkan untuk jejak layanan Lambda.
6. Pilih Simpan.

Mengaktifkan penelusuran dengan Lambda API

Konfigurasikan penelusuran pada fungsi Lambda Anda dengan AWS CLI AWS atau SDK, gunakan operasi API berikut:

- UpdateFunctionConfiguration
- GetFunctionConfiguration
- CreateFunction

Contoh AWS CLI perintah berikut memungkinkan penelusuran aktif pada fungsi bernama my-function.

```
aws lambda update-function-configuration --function-name my-function \
--tracing-config Mode=Active
```

Mode penelusuran adalah bagian dari konfigurasi khusus versi saat Anda memublikasikan versi fungsi Anda. Anda tidak dapat mengubah mode pelacakan pada versi yang telah diterbitkan.

Mengaktifkan penelusuran dengan CloudFormation

Untuk mengaktifkan penelusuran pada `AWS::Lambda::Function` sumber daya dalam CloudFormation templat, gunakan `TracingConfig` properti.

Example fungsi-inline.yml. — Konfigurasi penelusuran

```
Resources:
  function:
    Type: AWS::Lambda::Function
    Properties:
      TracingConfig:
        Mode: Active
      ...
```

Untuk sumber `AWS::Serverless::Function` daya AWS Serverless Application Model (AWS SAM), gunakan `Tracing` properti.

Example template.yml — Konfigurasi penelusuran

```
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      Tracing: Active
      ...
```

Menafsirkan jejak X-Ray

Fungsi Acara Anda memerlukan izin untuk mengunggah data jejak ke X-Ray. Saat Anda mengaktifkan penelusuran di konsol Lambda, Lambda menambahkan izin yang diperlukan ke peran eksekusi Fungsi Acara Anda. Atau, tambahkan kebijakan AWSXRayDaemonWriteAccess ke peran eksekusi.

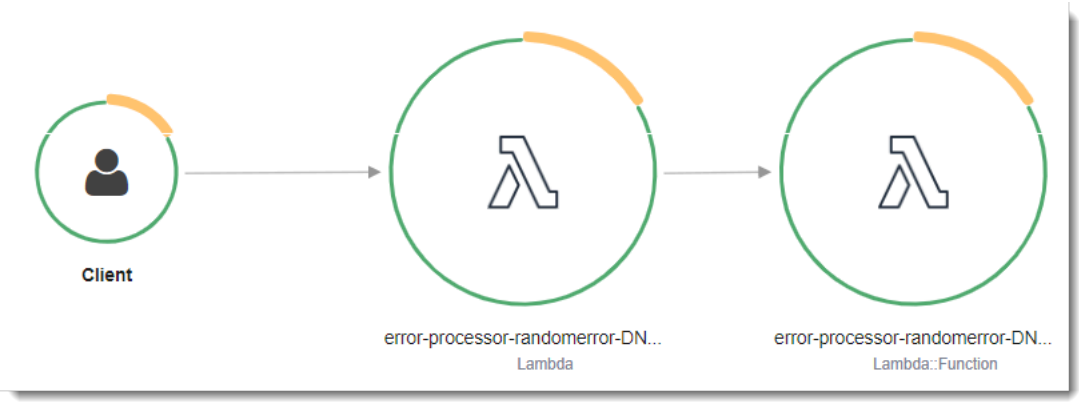
Setelah mengonfigurasi penelusuran aktif, Anda dapat mengamati permintaan tertentu melalui aplikasi Anda. Grafik X-Ray layanan menunjukkan informasi tentang aplikasi Anda dan semua komponennya. Contoh berikut menunjukkan aplikasi dengan dua fungsi. Fungsi utama memproses kejadian dan terkadang mengembalikan kesalahan. Fungsi kedua di bagian atas memproses

kesalahan yang muncul di grup log pertama dan menggunakan AWS SDK untuk memanggil X-Ray, Amazon Simple Storage Service (Amazon S3) Service, dan Amazon Logs. CloudWatch



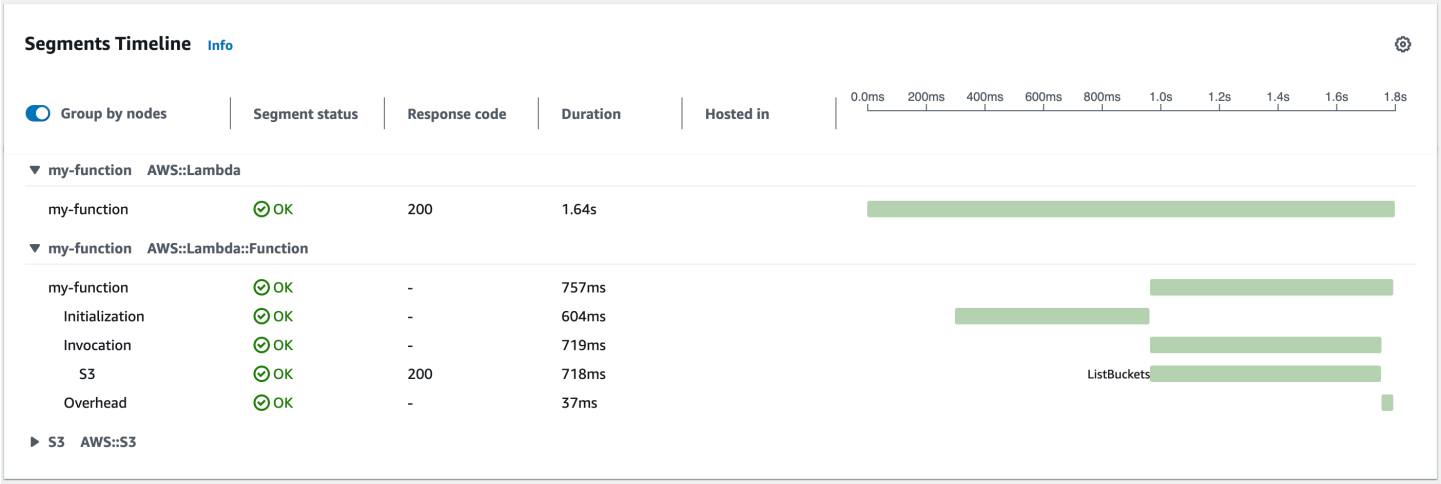
X-Ray tidak melacak semua permintaan ke aplikasi Anda. X-Ray menerapkan algoritma pengambilan sampel untuk memastikan bahwa penelusuran efisien, sambil tetap memberikan sampel yang representatif dari semua permintaan. Tingkat pengambilan sampel adalah 1 permintaan per detik dan 5 persen dari permintaan tambahan. Anda tidak dapat mengonfigurasi laju X-Ray pengambilan sampel untuk fungsi Anda.

Di X-Ray, jejak mencatat informasi tentang permintaan yang diproses oleh satu atau beberapa layanan. Lambda mencatat 2 segmen per jejak, yang menciptakan dua node pada grafik layanan. Gambar berikut menyoroti dua node ini:



Node pertama di sebelah kiri mewakili layanan Lambda, yang menerima permintaan pemanggilan. Node kedua mewakili fungsi Lambda spesifik Anda. Contoh berikut menunjukkan jejak dengan dua segmen ini. Keduanya bernama fungsi saya, tetapi yang satu memiliki asal `AWS::Lambda` dan yang lainnya memiliki asal usul. `AWS::Lambda::Function` Jika `AWS::Lambda` segmen

menunjukkan kesalahan, layanan Lambda mengalami masalah. Jika `AWS::Lambda::Function` segmen menunjukkan kesalahan, fungsi Anda mengalami masalah.



Contoh ini memperluas `AWS::Lambda::Function` segmen untuk menunjukkan tiga subsegmennya.

> **Note**
>
> AWS saat ini menerapkan perubahan pada layanan Lambda. Karena perubahan ini, Anda mungkin melihat perbedaan kecil antara struktur dan konten pesan log sistem dan segmen pelacakan yang dipancarkan oleh Fungsi Peristiwa Lambda yang berbeda di Anda. Akun AWS
> Contoh jejak yang ditunjukkan di sini menggambarkan segmen fungsi gaya lama. Perbedaan antara segmen gaya lama dan baru dijelaskan dalam paragraf berikut.
> Perubahan ini akan diterapkan selama beberapa minggu mendatang, dan semua fungsi di semua Wilayah AWS kecuali China GovCloud dan wilayah akan bertransisi untuk menggunakan pesan log format baru dan segmen pelacakan.

Segmen fungsi gaya lama berisi subsegmen berikut:

- Inisialisasi – Mewakili waktu yang dihabiskan untuk memuat fungsi dan menjalankan kode inisialisasi. Subsegmen ini hanya muncul untuk peristiwa pertama yang diproses oleh setiap instance fungsi Anda.
- Doa - Merupakan waktu yang dihabiskan untuk menjalankan kode handler Anda.
- Overhead - Merupakan waktu yang dihabiskan runtime Lambda untuk mempersiapkan diri untuk menangani acara berikutnya.

Segmen fungsi gaya baru tidak berisi subsegmen. `Invocation` Sebagai gantinya, subsegmen pelanggan dilampirkan langsung ke segmen fungsi. Untuk informasi lebih lanjut tentang struktur segmen fungsi gaya lama dan baru, lihat. the section called “Memahami X-Ray jejak”

> **Note**
>
> SnapStartfungsi juga termasuk `Restore` subsegmen. `Restore`Subsegmen menunjukkan waktu yang dibutuhkan Lambda untuk memulihkan snapshot, memuat runtime, dan menjalankan kait runtime setelah pemulihan. Proses memulihkan snapshot dapat mencakup waktu yang dihabiskan untuk aktivitas di luar microVM. Kali ini dilaporkan di `Restore` subsegmen. Anda tidak dikenakan biaya untuk waktu yang dihabiskan di luar microVM untuk memulihkan snapshot.

Anda juga dapat melakukan instrumentasi klien HTTP, merekam kueri SQL, dan membuat subsegmen khusus dengan anotasi dan metadata. Untuk informasi lebih lanjut, lihat AWS X-Ray SDK for Java dalam Panduan Pengembang AWS X-Ray .

> **Harga**
>
> Anda dapat menggunakan X-Ray tracing gratis setiap bulan hingga batas tertentu sebagai bagian dari Tingkat AWS Gratis. Di luar ambang batas itu, X-Ray biaya untuk penyimpanan dan pengambilan jejak. Untuk informasi selengkapnya, lihat harga AWS X-Ray.

Menyimpan dependensi runtime dalam lapisan (SDK) X-Ray

Jika Anda menggunakan X-Ray SDK untuk menginstruksikan klien AWS SDK kode fungsi Anda, paket penerapan Anda bisa menjadi cukup besar. Untuk menghindari mengunggah dependensi runtime setiap kali Anda memperbarui kode fungsi, paketkan X-Ray SDK dalam lapisan Lambda.

Contoh berikut menunjukkan `AWS::Serverless::LayerVersion` sumber daya yang menyimpan AWS SDK untuk Java dan X-Ray SDK for Java.

Example template.yml - Lapisan Dependensi

```
Resources:
  function:
    Type: AWS::Serverless::Function
```

```
    Properties:
      CodeUri: build/distributions/blank-java.zip
      Tracing: Active
      Layers:
        - !Ref libs
      ...
  libs:
    Type: AWS::Serverless::LayerVersion
    Properties:
      LayerName: blank-java-lib
      Description: Dependencies for the blank-java sample app.
      ContentUri: build/blank-java-lib.zip
      CompatibleRuntimes:
        - java25
```

Dengan konfigurasi ini, Anda memperbarui lapisan pustaka hanya jika Anda mengubah dependensi runtime Anda. Karena paket penerapan fungsi hanya berisi kode Anda, ini dapat membantu mengurangi waktu upload.

Membuat lapisan untuk dependensi memerlukan perubahan konfigurasi build untuk menghasilkan arsip lapisan sebelum penerapan. Untuk contoh kerja, lihat aplikasi sampel java-basic di. GitHub

X-Ray penelusuran dalam aplikasi sampel (X-Ray SDK)

GitHub Repositori untuk panduan ini mencakup contoh aplikasi yang menunjukkan penggunaan X-Ray tracing. Setiap contoh aplikasi menyertakan skrip untuk penyebaran dan pembersihan yang mudah, AWS SAM templat, dan sumber daya pendukung.

Sampel aplikasi Lambda di Java

- example-java - Fungsi Acara Java yang menunjukkan bagaimana Anda dapat menggunakan Lambda untuk memproses pesanan. Fungsi ini menggambarkan cara mendefinisikan dan deserialisasi objek peristiwa masukan kustom, menggunakan AWS SDK, dan logging keluaran.
- java-basic - Kumpulan fungsi Java minimal dengan pengujian unit dan konfigurasi logging variabel.
- java-events - Kumpulan fungsi Java yang berisi kode kerangka untuk cara menangani peristiwa dari berbagai layanan seperti Amazon API Gateway, Amazon SQS, dan Amazon Kinesis. Fungsi-fungsi ini menggunakan versi terbaru dari pustaka aws-lambda-java-events (3.0.0 dan yang lebih baru). Contoh-contoh ini tidak memerlukan AWS SDK sebagai dependensi.
- s3-java – Fungsi Java yang memproses kejadian pemberitahuan dari Amazon S3 dan menggunakan Java Class Library (JCL) untuk membuat thumbnail dari file gambar yang diunggah.

- layer-java — Fungsi Java yang menggambarkan cara menggunakan lapisan Lambda untuk mengemas dependensi yang terpisah dari kode fungsi inti Anda.

Semua aplikasi sampel memiliki pelacakan aktif yang diaktifkan untuk fungsi Lambda. Misalnya, `s3-java` aplikasi menampilkan instrumentasi otomatis AWS SDK for Java 2.x klien, manajemen segmen untuk pengujian, subsegmen khusus, dan penggunaan lapisan Lambda untuk menyimpan dependensi runtime.

Contoh aplikasi Java untuk AWS Lambda

GitHub Repositori untuk panduan ini menyediakan contoh aplikasi yang menunjukkan penggunaan Java di. AWS Lambda Setiap contoh aplikasi menyertakan skrip untuk penyebaran dan pembersihan yang mudah, CloudFormation templat, dan sumber daya pendukung.

Sampel aplikasi Lambda di Java

- example-java - Fungsi Acara Java yang menunjukkan bagaimana Anda dapat menggunakan Lambda untuk memproses pesanan. Fungsi ini menggambarkan cara mendefinisikan dan deserialisasi objek peristiwa input kustom, menggunakan AWS SDK, dan logging keluaran.
- java-basic - Kumpulan fungsi Java minimal dengan pengujian unit dan konfigurasi logging variabel.
- java-events - Kumpulan fungsi Java yang berisi kode kerangka untuk cara menangani peristiwa dari berbagai layanan seperti Amazon API Gateway, Amazon SQS, dan Amazon Kinesis. Fungsi-fungsi ini menggunakan versi terbaru dari pustaka aws-lambda-java-events (3.0.0 dan yang lebih baru). Contoh-contoh ini tidak memerlukan AWS SDK sebagai dependensi.
- s3-java – Fungsi Java yang memproses kejadian pemberitahuan dari Amazon S3 dan menggunakan Java Class Library (JCL) untuk membuat thumbnail dari file gambar yang diunggah.
- layer-java — Fungsi Java yang menggambarkan cara menggunakan lapisan Lambda untuk mengemas dependensi yang terpisah dari kode fungsi inti Anda.

Menjalankan kerangka kerja Java populer di Lambda

- spring-cloud-function-samples — Contoh dari Spring yang menunjukkan cara menggunakan framework Spring Cloud Function untuk membuat Lambda Event Functions. AWS
- Demo Aplikasi Boot Spring Tanpa Server - Contoh yang menunjukkan cara mengatur aplikasi Spring Boot khas dalam runtime Java yang dikelola dengan dan tanpa SnapStart, atau sebagai gambar asli GraalVM dengan runtime khusus.
- Demo Aplikasi Micronaut Tanpa Server - Contoh yang menunjukkan cara menggunakan Micronaut dalam runtime Java yang dikelola dengan dan tanpa SnapStart, atau sebagai gambar asli GraalVM dengan runtime kustom. Pelajari lebih lanjut di Micronaut/Lambda panduan.
- Demo Aplikasi Quarkus Tanpa Server - Contoh yang menunjukkan cara menggunakan Quarkus dalam runtime Java yang dikelola dengan dan tanpa SnapStart, atau sebagai gambar asli GraalVM dengan runtime kustom. Pelajari lebih lanjut di Quarkus/Lambda panduan dan Quarkus/SnapStart panduan.

Jika Anda baru mengenal Lambda Event Functions di Java, mulailah dengan contoh. `java-basic` Untuk memulai dengan sumber acara Lambda, lihat contohnya. `java-events` Kedua set contoh ini menunjukkan penggunaan pustaka Java Lambda, variabel lingkungan, AWS SDK, dan SDK. AWS X-Ray Contoh-contoh ini memerlukan pengaturan minimal dan Anda dapat menerapkannya dari baris perintah dalam waktu kurang dari satu menit.

Membangun Fungsi Acara Lambda dengan Go

Go diimplementasikan secara berbeda dari runtime terkelola lainnya. Karena Go mengkompilasi secara asli ke biner yang dapat dieksekusi, itu tidak memerlukan runtime bahasa khusus. Gunakan OS-only runtime (`provided` runtime family) untuk menerapkan Go Event Functions ke Lambda.

Untuk informasi tentang menggunakan layer dengan Go, lihat the section called “Lapisan”.

Topik

- Dukungan Go runtime
- Alat dan pustaka
- Tentukan penangan Fungsi Acara Lambda di Go
- Menggunakan objek konteks Lambda untuk mengambil informasi Go Event Function
- Terapkan Fungsi Acara Go Lambda dengan arsip file.zip
- Terapkan Fungsi Acara Go Lambda dengan gambar kontainer
- Bekerja dengan lapisan untuk Fungsi Acara Go Lambda
- Log dan pantau Fungsi Acara Go Lambda
- Menginstrumentasi kode Go di AWS Lambda

Dukungan Go runtime

Runtime terkelola Go 1.x untuk Lambda adalah tidak digunakan lagi. Jika Anda memiliki fungsi yang menggunakan runtime Go 1.x, Anda harus memigrasikan fungsi Anda ke `provided.al2023` atau `provided.al2`. `provided.al2023` dan `provided.al2` runtime menawarkan beberapa keuntungan di atas `go1.x`, termasuk dukungan untuk arsitektur arm64 (AWS Prosesor Graviton), biner yang lebih kecil, dan waktu pemanggilan sedikit lebih cepat.

Tidak ada perubahan kode yang diperlukan untuk migrasi ini. Satu-satunya perubahan yang diperlukan terkait dengan cara Anda membangun paket penerapan dan runtime mana yang Anda

gunakan untuk membuat fungsi Anda. Untuk informasi lebih lanjut, lihat Bermigrasi AWS Lambda fungsi dari Go1.x runtime ke runtime khusus di Amazon Linux 2 pada AWS Komputasi Blog.

Nama	Pengidentifikasi	Sistem operasi	Tanggal pengusangan	Blokir fungsi buat	Pembaruan fungsi blok
OS-only Waktu pelaksanaan	`provided.al2023`	Amazon Linux 2023	30 Juni 2029	31 Juli 2029	31 Agu 2029

Alat dan pustaka

Lambda menyediakan alat dan pustaka berikut untuk runtime Go:

- AWS SDK untuk Go v2: Pejabat AWS SDK untuk bahasa pemrograman Go.
- github.com/aws/aws-lambda-go/lambda: Implementasi model pemrograman Lambda untuk Go. Paket ini digunakan oleh AWS Lambda untuk memanggil Anda pengurus.
- github.com/aws/aws-lambda-go/lambdacontext: Pembantu untuk mengakses informasi konteks dari objek konteks.
- github.com/aws/aws-lambda-go/events: Pustaka ini menyediakan definisi tipe untuk integrasi sumber peristiwa umum.
- github.com/aws/aws-lambda-go/cmd/build-lambda-zip: Alat ini dapat digunakan untuk membuat arsip file.zip di Windows.

Untuk informasi lebih lanjut, lihat aws-lambda-go di atas GitHub.

Lambda menyediakan aplikasi contoh berikut untuk runtime Go:

Sampel aplikasi Lambda di Go

- go-al2 — Fungsi hello world yang mengembalikan alamat IP publik. Aplikasi ini menggunakan `provided.al2` runtime khusus.
- blank-go - Fungsi Go yang menunjukkan penggunaan perpustakaan Go Lambda, logging, variabel lingkungan, dan AWS SDK. Aplikasi ini menggunakan `go1.x` waktu pelaksanaan.

Tentukan penangan Fungsi Acara Lambda di Go

Lambda Event Function handler adalah metode dalam kode fungsi Anda yang memproses peristiwa. Saat fungsi Anda diaktifkan, Lambda menjalankan metode handler. Fungsi Anda berjalan sampai handler mengembalikan respons, keluar, atau waktu habis.

Halaman ini menjelaskan cara bekerja dengan penangan Fungsi Peristiwa Lambda di Go, termasuk penyiapan proyek, konvensi penamaan, dan praktik terbaik. Halaman ini juga menyertakan contoh fungsi Go Lambda yang mengambil informasi tentang pesanan, menghasilkan tanda terima file teks, dan menempatkan file ini di bucket Amazon Simple Storage Service (Amazon S3). Untuk informasi tentang cara menerapkan fungsi Anda setelah menulisnya, lihat the section called “Deploy arsip file .zip” atauthe section called “Deploy gambar kontainer”.

Topik

- Menyiapkan proyek Go handler Anda
- Contoh Kode Fungsi Acara Go Lambda
- Konvensi penamaan handler
- Mendefinisikan dan mengakses objek peristiwa masukan
- Mengakses dan menggunakan objek konteks Lambda
- Tanda tangan handler yang valid untuk penangan Go
- Menggunakan AWS SDK untuk Go v2 di handler Anda
- Mengakses variabel lingkungan
- Menggunakan status global
- Praktik terbaik kode untuk Fungsi Acara Go Lambda

Menyiapkan proyek Go handler Anda

Fungsi Lambda yang ditulis di Go diprogram sebagai Go executable. Anda dapat menginisialisasi proyek fungsi Go Lambda dengan cara yang sama Anda menginisialisasi proyek Go lainnya menggunakan perintah berikut: `go mod init`

```
go mod init example-go
```

Di sini, `example-go` adalah nama modul. Anda dapat menggantinya dengan apa saja. Perintah ini menginisialisasi proyek Anda dan menghasilkan `go.mod` file yang mencantumkan dependensi proyek Anda.

Gunakan `go get` perintah untuk menambahkan dependensi eksternal ke proyek Anda. Misalnya, untuk semua fungsi Lambda di Go, Anda harus menyertakan github. com/aws/aws-lambda- go/lambda paket, yang mengimplementasikan model pemrograman Lambda untuk Go. Sertakan paket ini dengan `go get` perintah berikut:

```
go get github.com/aws/aws-lambda-go
```

Kode fungsi Anda harus hidup dalam file Go. Dalam contoh berikut, kami memberi nama file ini`main.go`. Dalam file ini, Anda menerapkan logika fungsi inti Anda dalam metode handler, serta `main()` fungsi yang memanggil handler ini.

Contoh Kode Fungsi Acara Go Lambda

Contoh kode fungsi Go Lambda berikut mengambil informasi tentang pesanan, menghasilkan tanda terima file teks, dan menempatkan file ini di bucket Amazon S3.

Example **`main.go` Fungsi Acara Lambda**

```
package main

import (
 "context"
 "encoding/json"
 "fmt"
 "log"
 "os"
 "strings"

 "github.com/aws/aws-lambda-go/lambda"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/s3"
)

type Order struct {
 OrderID string  `json:"order_id"`
 Amount  float64 `json:"amount"`
 Item    string  `json:"item"`
}

var (
 s3Client *s3.Client
)
```

```
func init() {
 // Initialize the S3 client outside of the handler, during the init phase
 cfg, err := config.LoadDefaultConfig(context.TODO())
 if err != nil {
  log.Fatalf("unable to load SDK config, %v", err)
 }

 s3Client = s3.NewFromConfig(cfg)
}

func uploadReceiptToS3(ctx context.Context, bucketName, key, receiptContent string)
 error {
 _, err := s3Client.PutObject(ctx, &s3.PutObjectInput{
  Bucket: &bucketName,
  Key:    &key,
  Body:   strings.NewReader(receiptContent),
 })
 if err != nil {
  log.Printf("Failed to upload receipt to S3: %v", err)
  return err
 }
 return nil
}

func handleRequest(ctx context.Context, event json.RawMessage) error {
 // Parse the input event
 var order Order
 if err := json.Unmarshal(event, &order); err != nil {
  log.Printf("Failed to unmarshal event: %v", err)
  return err
 }

 // Access environment variables
 bucketName := os.Getenv("RECEIPT_BUCKET")
 if bucketName == "" {
  log.Printf("RECEIPT_BUCKET environment variable is not set")
  return fmt.Errorf("missing required environment variable RECEIPT_BUCKET")
 }

 // Create the receipt content and key destination
 receiptContent := fmt.Sprintf("OrderID: %s\nAmount: $%.2f\nItem: %s",
  order.OrderID, order.Amount, order.Item)
 key := "receipts/" + order.OrderID + ".txt"
```

```
 // Upload the receipt to S3 using the helper method
 if err := uploadReceiptToS3(ctx, bucketName, key, receiptContent); err != nil {
  return err
 }

 log.Printf("Successfully processed order %s and stored receipt in S3 bucket %s",
 order.OrderID, bucketName)
 return nil
}

func main() {
 lambda.Start(handleRequest)
}
```

`main.go`File ini berisi bagian kode berikut:

- `package main`: Di Go, paket yang berisi `func main()` fungsi Anda harus selalu diberi nama`main`.
- `import`block: Gunakan blok ini untuk menyertakan pustaka yang dibutuhkan oleh fungsi Lambda Anda.
- `type Order struct {}`block: Tentukan bentuk peristiwa input yang diharapkan dalam struct Go ini.
- `var ()`block: Gunakan blok ini untuk menentukan variabel global apa pun yang akan Anda gunakan dalam fungsi Lambda Anda.
- `func init() {}`: Sertakan kode apa pun yang Anda ingin Lambda jalankan selama fase inisialisasi dalam metode ini. `init()`
- `func uploadReceiptToS3(...) {}`: Ini adalah metode helper yang direferensikan oleh metode `handleRequest` handler utama.
- `func handleRequest(ctx context.Context, event json.RawMessage) error {}`: Ini adalah metode handler utama, yang berisi logika aplikasi utama Anda.
- `func main() {}`: Ini adalah titik masuk yang diperlukan untuk penangan Lambda Anda. Argumen untuk `lambda.Start()` metode ini adalah metode handler utama Anda.

Agar fungsi ini berfungsi dengan baik, peran pelaksanaannya harus memungkinkan `s3:PutObject` tindakan. Juga, pastikan bahwa Anda mendefinisikan variabel `RECEIPT_BUCKET` lingkungan. Setelah pemanggilan berhasil, bucket Amazon S3 harus berisi file tanda terima.

Konvensi penamaan handler

Untuk fungsi Lambda di Go, Anda dapat menggunakan nama apa pun untuk handler. Dalam contoh ini, nama metode handler adalah`handleRequest`. Untuk mereferensikan nilai handler dalam kode Anda, Anda dapat menggunakan variabel `_HANDLER` lingkungan.

Untuk fungsi Go yang digunakan menggunakan paket.zip deployment, file yang dapat dieksekusi yang berisi kode fungsi Anda harus diberi nama. `bootstrap` Selain itu, `bootstrap` file harus berada di root file.zip. Untuk fungsi Go yang digunakan menggunakan gambar kontainer, Anda dapat menggunakan nama apa pun untuk file yang dapat dieksekusi.

Mendefinisikan dan mengakses objek peristiwa masukan

JSON adalah format input yang paling umum dan standar untuk fungsi Lambda. Dalam contoh ini, fungsi mengharapkan input yang mirip dengan yang berikut:

```
{
    "order_id": "12345",
    "amount": 199.99,
    "item": "Wireless Headphones"
}
```

Saat bekerja dengan fungsi Lambda di Go, Anda dapat menentukan bentuk peristiwa input yang diharapkan sebagai struct Go. Dalam contoh ini, kita mendefinisikan struct untuk mewakili`Order`:

```
type Order struct {
    OrderID string  `json:"order_id"`
    Amount  float64 `json:"amount"`
    Item    string  `json:"item"`
}
```

Struct ini cocok dengan bentuk input yang diharapkan. Setelah Anda menentukan struct Anda, Anda dapat menulis tanda tangan handler yang menggunakan tipe JSON generik yang kompatibel dengan pustaka standar. encoding/json Anda kemudian dapat deserialisasinya ke dalam struct Anda menggunakan fungsi func Unmarshal. Ini diilustrasikan dalam beberapa baris pertama handler:

```
func handleRequest(ctx context.Context, event json.RawMessage) error {
    // Parse the input event
    var order Order
    if err := json.Unmarshal(event, &order); err != nil {
```

```
        log.Printf("Failed to unmarshal event: %v", err)
        return err
    ...
}
```

Setelah deserialisasi ini, Anda dapat mengakses bidang variabel. `order` Misalnya, `order.OrderID` mengambil nilai dari `"order_id"` dari input asli.

> **Note**
>
> `encoding/json`Paket hanya dapat mengakses bidang yang diekspor. Untuk diekspor, nama bidang dalam struktur peristiwa harus dalam huruf kapital.

Mengakses dan menggunakan objek konteks Lambda

Objek konteks Lambda berisi informasi tentang lingkungan pemanggilan, fungsi, dan eksekusi. Dalam contoh ini, kami mendeklarasikan variabel ini seperti `ctx` pada tanda tangan handler:

```
func handleRequest(ctx context.Context, event json.RawMessage) error {
    ...
}
```

`ctx context.Context`Input adalah argumen opsional dalam handler fungsi Anda. Untuk informasi selengkapnya tentang tanda tangan handler yang diterima, lihat. the section called “Tanda tangan handler yang valid untuk penangan Go”

Jika Anda melakukan panggilan ke layanan lain menggunakan AWS SDK, objek konteks diperlukan di beberapa area utama. Misalnya, untuk menginisialisasi klien SDK dengan benar, Anda dapat memuat konfigurasi AWS SDK yang benar menggunakan objek konteks sebagai berikut:

```
// Load AWS SDK configuration using the default credential provider chain
    cfg, err := config.LoadDefaultConfig(ctx)
```

Panggilan SDK sendiri mungkin memerlukan objek konteks sebagai input. Misalnya, `s3Client.PutObject` panggilan menerima objek konteks sebagai argumen pertamanya:

```
// Upload the receipt to S3
    _, err = s3Client.PutObject(ctx, &s3.PutObjectInput{
```

```
        ...
    })
```

Di luar permintaan AWS SDK, Anda juga dapat menggunakan objek konteks untuk pemantauan fungsi. Untuk informasi selengkapnya tentang objek konteks, lihatthe section called “Konteks”.

Tanda tangan handler yang valid untuk penangan Go

Anda memiliki beberapa pilihan ketika membangun handler fungsi Lambda di Go, tetapi Anda harus mematuhi aturan berikut:

- Handler harus berupa fungsi.
- Handler dapat mengambil antara 0 dan 2 argumen. Jika ada dua argumen, argumen pertama harus menerapkan `context.Context`.
- Handler dapat mengembalikan antara 0 dan 2 argumen. Jika ada nilai kembali tunggal, nilai harus menerapkan `error`. Jika ada dua nilai kembali, nilai kedua harus menerapkan `error`.

Berikut ini mencantumkan tanda tangan handler yang valid. `TIn`dan `TOut` mewakili tipe yang kompatibel dengan pustaka encoding/jsonstandar. Untuk informasi selengkapnya, lihat func Unmarshal untuk mempelajari cara tipe-tipe ini dideserialisasikan.

- `func ()`
- `func () error`
- `func () (TOut, error)`
- `func (TIn) error`
- `func (TIn) (TOut, error)`
- `func (context.Context) error`
- `func (context.Context) (TOut, error)`
- `func (context.Context, TIn) error`

- ```
  func (context.Context, TIn) (TOut, error)
  ```

Menggunakan AWS SDK untuk Go v2 di handler Anda

Seringkali, Anda akan menggunakan fungsi Lambda untuk berinteraksi dengan atau membuat pembaruan ke sumber daya lain AWS . Cara termudah untuk berinteraksi dengan sumber daya ini adalah dengan menggunakan AWS SDK untuk Go v2.

> **Note**
>
> AWS SDK untuk Go (v1) dalam mode pemeliharaan, dan akan mencapai akhir dukungan pada 31 Juli 2025. Kami menyarankan Anda hanya menggunakan AWS SDK untuk Go v2 ke depan.

Untuk menambahkan dependensi SDK ke fungsi Anda, gunakan `go get` perintah untuk klien SDK tertentu yang Anda butuhkan. Dalam contoh kode sebelumnya, kita menggunakan `config` perpustakaan dan `s3` perpustakaan. Tambahkan dependensi ini dengan menjalankan perintah berikut di direktori yang berisi file Anda `go.mod` dan`main.go` :

```
go get github.com/aws/aws-sdk-go-v2/config
go get github.com/aws/aws-sdk-go-v2/service/s3
```

Kemudian, impor dependensi yang sesuai di blok impor fungsi Anda:

```
import (
    ...
    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/service/s3"
)
```

Saat menggunakan SDK di handler Anda, konfigurasikan klien Anda dengan pengaturan yang tepat. Cara termudah untuk melakukannya adalah dengan menggunakan rantai penyedia kredensyal default. Contoh ini menggambarkan salah satu cara untuk memuat konfigurasi ini:

```
// Load AWS SDK configuration using the default credential provider chain
    cfg, err := config.LoadDefaultConfig(ctx)
    if err != nil {
```
```

```
        log.Printf("Failed to load AWS SDK config: %v", err)
        return err
    }
```

Setelah memuat konfigurasi ini ke dalam `cfg` variabel, Anda dapat meneruskan variabel ini ke instantiasi klien. Kode contoh membuat klien Amazon S3 sebagai berikut:

```
// Create an S3 client
    s3Client := s3.NewFromConfig(cfg)
```

Dalam contoh ini, kami menginisialisasi klien Amazon S3 kami dalam fungsi untuk menghindari `init()` keharusan menginisialisasi setiap kali kami menjalankan fungsi kami. Masalahnya adalah bahwa dalam `init()` fungsi, Lambda tidak memiliki akses ke objek konteks. Sebagai solusinya, Anda dapat meneruskan placeholder seperti `context.TODO()` selama fase inisialisasi. Kemudian, saat Anda melakukan panggilan menggunakan klien, teruskan objek konteks lengkap. Solusi ini juga dijelaskan dalam. the section called “Menggunakan konteks di AWS Inisialisasi dan panggilan klien SDK”

Setelah Anda mengkonfigurasi dan menginisialisasi klien SDK Anda, Anda kemudian dapat menggunakannya untuk berinteraksi dengan layanan lain AWS . Kode contoh memanggil Amazon S3 `PutObject` API sebagai berikut:

```
_, err = s3Client.PutObject(ctx, &s3.PutObjectInput{
    Bucket: &bucketName,
    Key:    &key,
    Body:   strings.NewReader(receiptContent),
})
```

Mengakses variabel lingkungan

Dalam kode handler Anda, Anda dapat mereferensikan variabel lingkungan apa pun dengan menggunakan `os.Getenv()` metode ini. Dalam contoh ini, kami mereferensikan variabel `RECEIPT_BUCKET` lingkungan yang ditentukan menggunakan baris kode berikut:

```
// Access environment variables
    bucketName := os.Getenv("RECEIPT_BUCKET")
    if bucketName == "" {
        log.Printf("RECEIPT_BUCKET environment variable is not set")
        return fmt.Errorf("missing required environment variable RECEIPT_BUCKET")
    }
```

Menggunakan status global

Untuk menghindari pembuatan sumber daya baru setiap kali Anda menjalankan fungsi, Anda dapat mendeklarasikan dan memodifikasi variabel global di luar kode penangan fungsi Lambda Anda. Anda mendefinisikan variabel global ini dalam `var` blok atau pernyataan. Selain itu, handler Anda dapat mendeklarasikan `init()` fungsi yang dijalankan selama fase inisialisasi. `init`Metode ini berperilaku sama AWS Lambda seperti pada program Go standar.

Praktik terbaik kode untuk Fungsi Acara Go Lambda

Patuhi pedoman dalam daftar berikut untuk menggunakan praktik pengkodean terbaik saat membangun fungsi Lambda Anda:

- Pisahkan handler Lambda dari logika inti Anda. Dengan pendekatan ini, Anda dapat membuat fungsi yang lebih dapat diuji unit.
- Minimalkan kompleksitas dependensi Anda. Utamakan memilih kerangka kerja lebih sederhana yang cepat dimuat dalam memulai lingkungan eksekusi.
- Minimalkan ukuran paket penerapan Anda sesuai kebutuhan runtime-nya. Ini mengurangi jumlah waktu yang diperlukan untuk paket penerapan Anda untuk diunduh dan dibongkar sebelum pemanggilan.

Manfaatkan penggunaan kembali lingkungan eksekusi untuk meningkatkan kinerja fungsi Anda. Inisialisasi klien SDK dan koneksi basis data di luar fungsi handler, dan lakukan caching aset statis secara lokal di direktori `/tmp`. Invokasi selanjutnya yang diproses oleh instans yang sama dari fungsi Anda dapat menggunakan kembali sumber daya ini. Ini menghemat biaya dengan mengurangi waktu pengoperasian fungsi.

Untuk menghindari potensi kebocoran data di seluruh invokasi, jangan menggunakan lingkungan eksekusi untuk menyimpan data pengguna, peristiwa, atau informasi lainnya implikasi keamanan. Jika fungsi Anda bergantung pada status yang dapat disenyapkan yang tidak dapat disimpan dalam memori di dalam handler, pertimbangkan untuk membuat fungsi terpisah atau versi terpisah dari fungsi untuk setiap pengguna.

Gunakan arahan keep-alive untuk mempertahankan koneksi yang persisten. Lambda membersihkan koneksi idle dari waktu ke waktu. Mencoba menggunakan ulang koneksi idle saat mengidentifikasi suatu fungsi akan menyebabkan kesalahan koneksi. Untuk mempertahankan koneksi yang persisten, gunakan arahan tetap aktif yang berkaitan dengan runtime Anda. Sebagai contoh, lihat Menggunakan Kembali Koneksi dengan Keep-Alive in Node.js.

Gunakan variabel lingkungan untuk meneruskan parameter operasional ke fungsi Anda. Misalnya, jika Anda ingin menulis ke bucket Amazon S3 alih-alih melakukan hard-coding nama bucket yang Anda tulis, konfigurasikan nama bucket sebagai variabel lingkungan.

Hindari menggunakan pemanggilan rekursif dalam fungsi Lambda Anda, di mana fungsi memanggil dirinya sendiri atau memulai proses yang dapat memanggil fungsi lagi. Hal ini dapat menyebabkan volume invokasi fungsi yang tidak diinginkan dan peningkatan biaya. Jika Anda melihat volume pemanggilan yang tidak diinginkan, setel konkurensi fungsi cadangan untuk `0` segera membatasi semua pemanggilan ke fungsi, saat Anda memperbarui kode.

Jangan gunakan API non-publik yang tidak terdokumentasi dalam kode fungsi Lambda Anda. Untuk runtime AWS Lambda terkelola, Lambda secara berkala menerapkan pembaruan keamanan dan fungsional ke API internal Lambda. Pembaruan API internal ini mungkin tidak kompatibel ke belakang, yang menyebabkan konsekuensi yang tidak diinginkan seperti kegagalan pemanggilan jika fungsi Anda memiliki ketergantungan pada API non-publik ini. Lihat referensi API untuk daftar API yang tersedia untuk umum.

Tulis kode idempoten. Menulis kode idempoten untuk fungsi Anda memastikan bahwa peristiwa duplikat ditangani dengan cara yang sama. Kode Anda harus memvalidasi peristiwa dengan benar dan menangani peristiwa duplikat dengan anggun. Untuk informasi selengkapnya, lihat Bagaimana cara membuat fungsi Lambda saya idempoten? .

Menggunakan objek konteks Lambda untuk mengambil informasi Go Event Function

Di Lambda, objek konteks menyediakan metode dan properti dengan informasi tentang lingkungan pemanggilan, fungsi, dan eksekusi. Saat Lambda menjalankan fungsi Anda, ia meneruskan objek konteks ke handler. Untuk menggunakan objek konteks di handler Anda, Anda dapat secara opsional mendeklarasikannya sebagai parameter input ke handler Anda. Objek konteks diperlukan jika Anda ingin melakukan hal berikut di handler Anda:

- Anda memerlukan akses ke salah satu variabel global, metode, atau properti yang ditawarkan oleh objek konteks. Metode dan properti ini berguna untuk tugas-tugas seperti menentukan entitas yang memanggil fungsi Anda atau mengukur waktu pemanggilan fungsi Anda, seperti yang diilustrasikan dalam. the section called “Mengakses informasi konteks aktif”
- Anda perlu menggunakan AWS SDK untuk Go untuk melakukan panggilan ke layanan lain. Objek konteks adalah parameter input penting untuk sebagian besar panggilan ini. Untuk informasi selengkapnya, lihat the section called “Menggunakan konteks di AWS Inisialisasi dan panggilan klien SDK”.

Topik

- Variabel, metode, dan properti yang didukung dalam objek konteks
- Mengakses informasi konteks aktif
- Menggunakan konteks di AWS Inisialisasi dan panggilan klien SDK

Variabel, metode, dan properti yang didukung dalam objek konteks

Pustaka konteks Lambda menyediakan variabel, metode, dan sifat global berikut.

Variabel global

- `FunctionName` – Nama fungsi Lambda.
- `FunctionVersion` – Versi fungsi.
- `MemoryLimitInMB` – Jumlah memori yang dialokasikan untuk fungsi tersebut.
- `LogGroupName` – Grup log untuk fungsi.
- `LogStreamName` – Aliran log untuk instans fungsi.

Metode konteks

- `Deadline` – Mengembalikan tanggal saat waktu pelaksanaan habis, dalam waktu Unix milidetik.

Properti konteks

- `InvokedFunctionArn` – Amazon Resource Name (ARN) yang digunakan untuk memicu fungsi. Menunjukkan jika pemicu menyebutkan nomor versi atau alias.
- `AwsRequestID` – Pengidentifikasi permintaan invokasi.
- `Identity` – (aplikasi seluler) Informasi tentang identitas Amazon Cognito yang mengesahkan permintaan.
- `ClientContext` – (aplikasi seluler) Konteks klien yang disediakan untuk Lambda oleh aplikasi klien.

Mengakses informasi konteks aktif

Fungsi Lambda memiliki akses ke metadata tentang lingkungan mereka dan permintaan invokasi. Ini dapat diakses di Konteks paket. Jika handler Anda mencakup `context.Context` sebagai parameter, Lambda akan memasukkan informasi tentang fungsi Anda ke dalam `Value` properti konteks. Perhatikan bahwa Anda perlu mengimpor pustaka `lambdacontext` untuk mengakses konten dari `context.Context` objek.

```
package main

import (
        "context"
        "log"
        "github.com/aws/aws-lambda-go/lambda"
        "github.com/aws/aws-lambda-go/lambdacontext"
)

func CognitoHandler(ctx context.Context) {
        lc, _ := lambdacontext.FromContext(ctx)
        log.Print(lc.Identity.CognitoIdentityPoolID)
}

func main() {
        lambda.Start(CognitoHandler)
}
```

Dalam contoh di atas, `lc` adalah variabel yang digunakan untuk menggunakan informasi yang ditangkap oleh objek konteks dan `log.Print(lc.Identity.CognitoIdentityPoolID)` mencetak informasi tersebut, dalam kasus ini, CognitoIdentityPoolID.

Contoh berikut memperkenalkan cara menggunakan objek konteks untuk memantau berapa lama waktu yang dibutuhkan fungsi Lambda untuk menyelesaikannya. Dengan cara ini, Anda dapat menganalisis ekspektasi kinerja dan menyesuaikan kode fungsi Anda sesuai, jika diperlukan.

```
package main

import (
        "context"
        "log"
        "time"
        "github.com/aws/aws-lambda-go/lambda"
)

func LongRunningHandler(ctx context.Context) (string, error) {

        deadline, _ := ctx.Deadline()
        deadline = deadline.Add(-100 * time.Millisecond)
        timeoutChannel := time.After(time.Until(deadline))

        for {

                select {

                case <- timeoutChannel:
                        return "Finished before timing out.", nil

                default:
                        log.Print("hello!")
                        time.Sleep(50 * time.Millisecond)
                }
        }
}

func main() {
        lambda.Start(LongRunningHandler)
}
```

Menggunakan konteks di AWS Inisialisasi dan panggilan klien SDK

Jika handler Anda perlu menggunakan untuk melakukan panggilan AWS SDK untuk Go ke layanan lain, sertakan objek konteks sebagai input ke handler Anda. Di AWS, ini adalah praktik terbaik untuk meneruskan objek konteks di sebagian besar panggilan AWS SDK. Misalnya, `PutObject` panggilan Amazon S3 menerima objek konteks (`ctx`) sebagai argumen pertamanya:

```
// Upload an object to S3
    _, err = s3Client.PutObject(ctx, &s3.PutObjectInput{
        ...
    })
```

Untuk menginisialisasi klien SDK Anda dengan benar, Anda juga dapat menggunakan objek konteks untuk memuat konfigurasi yang benar sebelum meneruskan objek konfigurasi tersebut ke klien:

```
// Load AWS SDK configuration using the default credential provider chain
    cfg, err := config.LoadDefaultConfig(ctx)
    ...
    s3Client = s3.NewFromConfig(cfg)
```

Jika Anda ingin menginisialisasi klien SDK di luar handler utama Anda (yaitu selama fase inisialisasi), Anda dapat meneruskan objek konteks placeholder:

```
func init() {
 // Initialize the S3 client outside of the handler, during the init phase
 cfg, err := config.LoadDefaultConfig(context.TODO())
 ...
 s3Client = s3.NewFromConfig(cfg)
}
```

Jika Anda menginisialisasi klien dengan cara ini, pastikan Anda meneruskan objek konteks yang benar dalam panggilan SDK dari pengendali utama Anda.

Terapkan Fungsi Acara Go Lambda dengan arsip file.zip

Kode AWS Lambda fungsi Anda terdiri dari skrip atau program yang dikompilasi dan dependensinya. Gunakan paket deployment untuk men-deploy fungsi kode Anda ke Lambda. Lambda mendukung dua tipe paket deployment: gambar kontainer dan arsip file .zip.

Halaman ini menjelaskan cara membuat file.zip sebagai paket penerapan Anda untuk runtime Go, dan kemudian menggunakan file.zip untuk menyebarkan kode fungsi Anda AWS Lambda menggunakan, () Konsol Manajemen AWS, dan AWS Command Line Interface (AWS CLI). AWS Serverless Application Model AWS SAM

Perhatikan bahwa Lambda menggunakan izin file POSIX, jadi Anda mungkin perlu mengatur izin untuk folder paket penyebaran sebelum membuat arsip file.zip.

Bagian-bagian

- Membuat file .zip pada macOS dan Linux
- Membuat file .zip pada Windows
- Membuat dan memperbarui Fungsi Acara Go Lambda menggunakan file.zip

Membuat file .zip pada macOS dan Linux

Langkah-langkah berikut menunjukkan cara mengkompilasi executable Anda menggunakan go `build` perintah dan membuat paket penyebaran file.zip untuk Lambda. Sebelum mengkompilasi kode Anda, pastikan Anda telah menginstal paket lambda dari. GitHub Modul ini menyediakan implementasi antarmuka runtime, yang mengelola interaksi antara Lambda dan kode fungsi Anda. Untuk mengunduh pustaka ini, jalankan perintah berikut.

```
go get github.com/aws/aws-lambda-go/lambda
```

Jika fungsi Anda menggunakan AWS SDK untuk Go, unduh set standar modul SDK, bersama dengan klien API AWS layanan apa pun yang diperlukan oleh aplikasi Anda. Untuk mempelajari cara menginstal SDK for Go, lihat Memulai dengan V2 AWS SDK untuk Go .

Menggunakan keluarga runtime yang disediakan

Go diimplementasikan secara berbeda dari runtime terkelola lainnya. Karena Go mengkompilasi secara native ke biner yang dapat dieksekusi, itu tidak memerlukan runtime bahasa khusus. Gunakan OS-only runtime (keluarga `provided` runtime) untuk menerapkan Go Event Functions ke Lambda.

Untuk membuat paket penyebaran .zip () macOS/Linux

1. Dalam direktori proyek yang berisi `main.go` file aplikasi Anda, kompilasi executable Anda. Perhatikan hal-hal berikut:

   - Executable harus diberi nama. `bootstrap` Untuk informasi selengkapnya, lihat Konvensi penamaan handler.
   - Tetapkan arsitektur set instruksi target Anda. OS-only runtime mendukung arm64 dan x86_64.
   - Anda dapat menggunakan `lambda.norpc` tag opsional untuk mengecualikan komponen Remote Procedure Call (RPC) dari pustaka lambda. Komponen RPC hanya diperlukan jika Anda menggunakan runtime Go 1.x yang tidak digunakan lagi. Mengecualikan RPC mengurangi ukuran paket penyebaran.

   Untuk arsitektur arm64:

   ```
   GOOS=linux GOARCH=arm64 go build -tags lambda.norpc -o bootstrap main.go
   ```

   Untuk arsitektur x86_64:

   ```
   GOOS=linux GOARCH=amd64 go build -tags lambda.norpc -o bootstrap main.go
   ```

2. (Opsional) Anda mungkin perlu mengompilasi paket dengan `CGO_ENABLED=0` yang diatur di Linux:

   ```
   GOOS=linux GOARCH=arm64 CGO_ENABLED=0 go build -o bootstrap -tags lambda.norpc
    main.go
   ```

   Perintah ini membuat paket biner stabil untuk versi pustaka C standar (`libc`), yang mungkin berbeda di Lambda dan perangkat lainnya.

3. Buat paket deployment dengan mengemas executable dalam file .zip.

   ```
   zip myFunction.zip bootstrap
   ```

   > **Note**
   >
   > `bootstrap`File harus berada di root file.zip.

4. Buat fungsi . Perhatikan hal-hal berikut:

   - Biner harus diberi namabootstrap, tetapi nama handler bisa apa saja. Untuk informasi selengkapnya, lihat Konvensi penamaan handler.
   - --architecturesOpsi ini hanya diperlukan jika Anda menggunakan arm64. Nilai default adalah x86_64.
   - Untuk--role, tentukan Nama Sumber Daya Amazon (ARN) dari peran eksekusi.

```
aws lambda create-function --function-name myFunction \
--runtime provided.al2023 --handler bootstrap \
--architectures arm64 \
--role arn:aws:iam::111122223333:role/lambda-ex \
--zip-file fileb://myFunction.zip
```

Membuat file .zip pada Windows

Langkah-langkah berikut menunjukkan cara mengunduh alat build-lambda-zip untuk Windows dari GitHub, mengkompilasi executable Anda, dan membuat paket deployment .zip.

> **Note**
>
> Jika Anda belum melakukannya, Anda harus menginstal git, lalu tambahkan executable `git` ke variabel lingkungan Windows %PATH% Anda.

Sebelum mengkompilasi kode Anda, pastikan Anda telah menginstal pustaka lambda dari. GitHub Untuk mengunduh pustaka ini, jalankan perintah berikut.

```
go get github.com/aws/aws-lambda-go/lambda
```

Jika fungsi Anda menggunakan AWS SDK untuk Go, unduh set standar modul SDK, bersama dengan klien API AWS layanan apa pun yang diperlukan oleh aplikasi Anda. Untuk mempelajari cara menginstal SDK for Go, lihat Memulai dengan V2 AWS SDK untuk Go .

Menggunakan keluarga runtime yang disediakan

Go diimplementasikan secara berbeda dari runtime terkelola lainnya. Karena Go mengkompilasi secara native ke biner yang dapat dieksekusi, itu tidak memerlukan runtime bahasa khusus. Gunakan OS-only runtime (keluarga `provided` runtime) untuk menerapkan Go Event Functions ke Lambda.

Untuk membuat paket penyebaran.zip (Windows)

1. Unduh alat build-lambda-zip dari. GitHub

   ```
   go install github.com/aws/aws-lambda-go/cmd/build-lambda-zip@latest
   ```

2. Gunakan alat dari `GOPATH` Anda untuk membuat file .zip. Jika Anda memiliki penginstalan default Go, alat ini biasanya di `%USERPROFILE%\Go\bin`. Jika tidak, arahkan ke tempat Anda menginstal runtime Go dan lakukan salah satu hal berikut:

   cmd.exe

   Di cmd.exe, jalankan salah satu dari berikut ini, tergantung pada arsitektur set instruksi target Anda. OS-only runtime mendukung arm64 dan x86_64.

   Anda dapat menggunakan `lambda.norpc` tag opsional untuk mengecualikan komponen Remote Procedure Call (RPC) dari pustaka lambda. Komponen RPC hanya diperlukan jika Anda menggunakan runtime Go 1.x yang tidak digunakan lagi. Mengecualikan RPC mengurangi ukuran paket penyebaran.

   Example— Untuk arsitektur x86_64

   ```
   set GOOS=linux
   set GOARCH=amd64
   set CGO_ENABLED=0
   go build -tags lambda.norpc -o bootstrap main.go
   %USERPROFILE%\Go\bin\build-lambda-zip.exe -o myFunction.zip bootstrap
   ```

   Example— Untuk arsitektur arm64

   ```
   set GOOS=linux
   set GOARCH=arm64
   set CGO_ENABLED=0
   go build -tags lambda.norpc -o bootstrap main.go
   ```

```
%USERPROFILE%\Go\bin\build-lambda-zip.exe -o myFunction.zip bootstrap
```

PowerShell

Dalam PowerShell, jalankan salah satu dari berikut ini, tergantung pada arsitektur set instruksi target Anda. OS-only runtime mendukung arm64 dan x86_64.

Anda dapat menggunakan `lambda.norpc` tag opsional untuk mengecualikan komponen Remote Procedure Call (RPC) dari pustaka lambda. Komponen RPC hanya diperlukan jika Anda menggunakan runtime Go 1.x yang tidak digunakan lagi. Mengecualikan RPC mengurangi ukuran paket penyebaran.

Untuk arsitektur x86_64:

```
$env:GOOS = "linux"
$env:GOARCH = "amd64"
$env:CGO_ENABLED = "0"
go build -tags lambda.norpc -o bootstrap main.go
~\Go\Bin\build-lambda-zip.exe -o myFunction.zip bootstrap
```

Untuk arsitektur arm64:

```
$env:GOOS = "linux"
$env:GOARCH = "arm64"
$env:CGO_ENABLED = "0"
go build -tags lambda.norpc -o bootstrap main.go
~\Go\Bin\build-lambda-zip.exe -o myFunction.zip bootstrap
```

3. Buat fungsi . Perhatikan hal-hal berikut:

- Biner harus diberi nama`bootstrap`, tetapi nama handler bisa apa saja. Untuk informasi selengkapnya, lihat Konvensi penamaan handler.
- `--architectures`Opsi ini hanya diperlukan jika Anda menggunakan arm64. Nilai default adalah x86_64.
- Untuk`--role`, tentukan Nama Sumber Daya Amazon (ARN) dari peran eksekusi.

```
aws lambda create-function --function-name myFunction \
--runtime provided.al2023 --handler bootstrap \
--architectures arm64 \
```

```
--role arn:aws:iam::111122223333:role/lambda-ex \
--zip-file fileb://myFunction.zip
```

Membuat dan memperbarui Fungsi Acara Go Lambda menggunakan file.zip

Setelah Anda membuat paket.zip deployment, Anda dapat menggunakannya untuk membuat fungsi Lambda baru atau memperbarui yang sudah ada. Anda dapat menerapkan paket.zip Anda menggunakan konsol Lambda, API, AWS Command Line Interface dan Lambda. Anda juga dapat membuat dan memperbarui fungsi Lambda menggunakan AWS Serverless Application Model (AWS SAM) dan. CloudFormation

Ukuran maksimum untuk paket.zip deployment untuk Lambda adalah 250 MB (unzip). Perhatikan bahwa batas ini berlaku untuk ukuran gabungan semua file yang Anda unggah, termasuk lapisan Lambda apa pun.

Runtime Lambda membutuhkan izin untuk membaca file dalam paket deployment Anda. Dalam notasi oktal izin Linux, Lambda membutuhkan 644 izin untuk file yang tidak dapat dieksekusi (rw-r--r--) dan 755 izin (rwxr-xr-x) untuk direktori dan file yang dapat dieksekusi.

Di Linux dan macOS, gunakan `chmod` perintah untuk mengubah izin file pada file dan direktori dalam paket penyebaran Anda. Misalnya, untuk memberikan file yang tidak dapat dieksekusi izin yang benar, jalankan perintah berikut.

```
chmod 644 <filepath>
```

Untuk mengubah izin file di Windows, lihat Mengatur, Melihat, Mengubah, atau Menghapus Izin pada Objek dalam dokumentasi Microsoft Windows.

> **Note**
>
> Jika Anda tidak memberikan Lambda izin yang diperlukan untuk mengakses direktori dalam paket penerapan Anda, Lambda menetapkan izin untuk direktori tersebut ke 755 (rwxr-xr-x).

Membuat dan memperbarui fungsi dengan file.zip menggunakan konsol

Untuk membuat fungsi baru, Anda harus terlebih dahulu membuat fungsi di konsol, lalu mengunggah arsip.zip Anda. Untuk memperbarui fungsi yang ada, buka halaman untuk fungsi Anda, lalu ikuti prosedur yang sama untuk menambahkan file.zip Anda yang diperbarui.

Jika file.zip Anda kurang dari 50MB, Anda dapat membuat atau memperbarui fungsi dengan mengunggah file langsung dari mesin lokal Anda. Untuk file.zip yang lebih besar dari 50MB, Anda harus mengunggah paket Anda ke bucket Amazon S3 terlebih dahulu. Untuk petunjuk tentang cara mengunggah file ke bucket Amazon S3 menggunakan Konsol Manajemen AWS, lihat Memulai Amazon S3. Untuk mengunggah file menggunakan AWS CLI, lihat Memindahkan objek di Panduan AWS CLI Pengguna.

> **Note**
>
> Anda tidak dapat mengonversi fungsi gambar kontainer yang ada untuk menggunakan arsip.zip. Anda harus membuat fungsi baru.

Untuk membuat fungsi baru (konsol)

1. Buka halaman Fungsi Acara konsol Lambda dan pilih Buat Fungsi Acara.
2. Pilih Tulis dari awal.
3. Di bagian Informasi dasar, lakukan hal berikut:
   a. Untuk nama Fungsi, masukkan nama untuk fungsi Anda.
   b. Untuk Runtime, pilih`provided.al2023`.
4. (Opsional) Di bagian Izin, luaskan Ubah peran eksekusi default. Anda dapat membuat peran Eksekusi baru atau menggunakan yang sudah ada.
5. Pilih Buat Fungsi Acara. Lambda menciptakan fungsi dasar 'Hello world' menggunakan runtime yang Anda pilih.

Untuk mengunggah arsip.zip dari mesin lokal Anda (konsol)

1. Di halaman Fungsi Acara konsol Lambda, pilih fungsi yang ingin Anda unggah file.zip.
2. Pilih tab Kode.
3. Di panel Sumber kode, pilih Unggah dari.
4. Pilih file.zip.
5. Untuk mengunggah file.zip, lakukan hal berikut:
   a. Pilih Unggah, lalu pilih file.zip Anda di pemilih file.
   b. Pilih Buka.

c. Pilih Simpan.

Untuk mengunggah arsip.zip dari bucket Amazon S3 (konsol)

1. Di halaman Fungsi Acara konsol Lambda, pilih fungsi yang ingin Anda unggah file.zip baru.
2. Pilih tab Kode.
3. Di panel Sumber kode, pilih Unggah dari.
4. Pilih lokasi Amazon S3.
5. Rekatkan URL tautan Amazon S3 dari file.zip Anda dan pilih Simpan.

Membuat dan memperbarui fungsi dengan file.zip menggunakan AWS CLI

Anda dapat menggunakan AWS CLIuntuk membuat fungsi baru atau memperbarui yang sudah ada menggunakan file.zip. Gunakan perintah create-function dan update-function-code untuk menyebarkan paket.zip Anda. Jika file.zip Anda lebih kecil dari 50MB, Anda dapat mengunggah paket.zip dari lokasi file di mesin build lokal Anda. Untuk file yang lebih besar, Anda harus mengunggah paket.zip Anda dari bucket Amazon S3. Untuk petunjuk tentang cara mengunggah file ke bucket Amazon S3 menggunakan AWS CLI, lihat Memindahkan objek di AWS CLI Panduan Pengguna.

> **Note**
>
> Jika Anda mengunggah file.zip dari bucket Amazon S3 menggunakan AWS CLI bucket, bucket harus berada di lokasi Wilayah AWS yang sama dengan fungsi Anda.

Untuk membuat fungsi baru menggunakan file.zip dengan AWS CLI, Anda harus menentukan yang berikut:

- Nama fungsi Anda (`--function-name`)
- Runtime () `--runtime` fungsi Anda
- Nama Sumber Daya Amazon (ARN) dari peran eksekusi fungsi Anda () `--role`
- Nama metode handler dalam kode fungsi Anda () `--handler`

Anda juga harus menentukan lokasi file.zip Anda. Jika file.zip Anda terletak di folder di mesin build lokal Anda, gunakan `--zip-file` opsi untuk menentukan jalur file, seperti yang ditunjukkan pada perintah contoh berikut.

```
aws lambda create-function --function-name myFunction \
--runtime provided.al2023 --handler bootstrap \
--role arn:aws:iam::111122223333:role/service-role/my-lambda-role \
--zip-file fileb://myFunction.zip
```

Untuk menentukan lokasi file.zip di bucket Amazon S3, gunakan opsi seperti `--code` yang ditunjukkan pada perintah contoh berikut. Anda hanya perlu menggunakan `S3ObjectVersion` parameter untuk objek berversi.

```
aws lambda create-function --function-name myFunction \
--runtime provided.al2023 --handler bootstrap \
--role arn:aws:iam::111122223333:role/service-role/my-lambda-role \
--code S3Bucket=amzn-s3-demo-
bucket,S3Key=myFileName.zip,S3ObjectVersion=myObjectVersion
```

Untuk memperbarui fungsi yang ada menggunakan CLI, Anda menentukan nama fungsi Anda menggunakan parameter. `--function-name` Anda juga harus menentukan lokasi file.zip yang ingin Anda gunakan untuk memperbarui kode fungsi Anda. Jika file.zip Anda terletak di folder di mesin build lokal Anda, gunakan `--zip-file` opsi untuk menentukan jalur file, seperti yang ditunjukkan pada perintah contoh berikut.

```
aws lambda update-function-code --function-name myFunction \
--zip-file fileb://myFunction.zip
```

Untuk menentukan lokasi file.zip di bucket Amazon S3, gunakan opsi `--s3-key` dan seperti `--s3-bucket` yang ditunjukkan pada perintah contoh berikut. Anda hanya perlu menggunakan `--s3-object-version` parameter untuk objek berversi.

```
aws lambda update-function-code --function-name myFunction \
--s3-bucket amzn-s3-demo-bucket --s3-key myFileName.zip --s3-object-version myObject
 Version
```

Membuat dan memperbarui fungsi dengan file.zip menggunakan API Lambda

Untuk membuat dan memperbarui fungsi menggunakan arsip file.zip, gunakan operasi API berikut:

- CreateFunction
- UpdateFunctionCode

Membuat dan memperbarui fungsi dengan file.zip menggunakan AWS SAM

The AWS Serverless Application Model (AWS SAM) adalah toolkit yang membantu merampingkan proses membangun dan menjalankan aplikasi tanpa server. AWS Anda menentukan sumber daya untuk aplikasi Anda dalam template YAMB atau JSON dan menggunakan antarmuka baris AWS SAM perintah (AWS SAM CLI) untuk membangun, mengemas, dan menyebarkan aplikasi Anda. Saat Anda membuat fungsi Lambda dari AWS SAM template, AWS SAM secara otomatis membuat paket penerapan .zip atau gambar kontainer dengan kode fungsi Anda dan dependensi apa pun yang Anda tentukan. Untuk mempelajari lebih lanjut cara menggunakan AWS SAM untuk membangun dan menerapkan fungsi Lambda, lihat Memulai di Panduan AWS SAMAWS Serverless Application Model Pengembang.

Anda juga dapat menggunakan AWS SAM untuk membuat fungsi Lambda menggunakan arsip file.zip yang ada. Untuk membuat fungsi Lambda menggunakan AWS SAM, Anda dapat menyimpan file.zip di bucket Amazon S3 atau di folder lokal di mesin build Anda. Untuk petunjuk tentang cara mengunggah file ke bucket Amazon S3 menggunakan AWS CLI, lihat Memindahkan objek di AWS CLI Panduan Pengguna.

Dalam AWS SAM template Anda, `AWS::Serverless::Function` sumber daya menentukan fungsi Lambda Anda. Dalam sumber daya ini, atur properti berikut untuk membuat fungsi menggunakan arsip file.zip:

- `PackageType`- diatur ke `Zip`
- `CodeUri`- diatur ke kode fungsi Amazon S3 URI, jalur ke folder lokal, atau objek FunctionCode
- `Runtime`- Setel ke runtime yang Anda pilih

Dengan AWS SAM, jika file.zip Anda lebih besar dari 50MB, Anda tidak perlu mengunggahnya ke bucket Amazon S3 terlebih dahulu. AWS SAM dapat mengunggah paket.zip hingga ukuran maksimum yang diizinkan 250MB (unzip) dari lokasi di mesin build lokal Anda.

Untuk mempelajari selengkapnya tentang penerapan fungsi menggunakan file.zip di AWS SAM, lihat AWS::Serverless: :Fungsi dalam Panduan PengembangAWS SAM .

Contoh: Menggunakan AWS SAM untuk membangun fungsi Go dengan provided.al2023

1. Buat AWS SAM template dengan properti berikut:

- BuildMethod: Menentukan compiler untuk aplikasi Anda. Gunakan `go1.x`.
- Runtime: Gunakan`provided.al2023`.
- CodeUri: Masukkan jalur ke kode Anda.
- Arsitektur: Gunakan `[arm64]` untuk arsitektur arm64. Untuk arsitektur set instruksi x86_64, gunakan `[amd64]` atau hapus properti. `Architectures`

Example template.yaml

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: 'AWS::Serverless-2016-10-31'
Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function
    Metadata:
      BuildMethod: go1.x
    Properties:
      CodeUri: hello-world/ # folder where your main program resides
      Handler: bootstrap
      Runtime: provided.al2023
      Architectures: [arm64]
```

2. Gunakan perintah sam build untuk mengkompilasi executable.

```
sam build
```

3. Gunakan perintah sam deploy untuk menyebarkan fungsi ke Lambda.

```
sam deploy --guided
```

Membuat dan memperbarui fungsi dengan file.zip menggunakan CloudFormation

Anda dapat menggunakan CloudFormation untuk membuat fungsi Lambda menggunakan arsip file.zip. Untuk membuat fungsi Lambda dari file.zip, Anda harus terlebih dahulu mengunggah file Anda ke bucket Amazon S3. Untuk petunjuk tentang cara mengunggah file ke bucket Amazon S3 menggunakan AWS CLI, lihat Memindahkan objek di AWS CLI Panduan Pengguna.

Dalam CloudFormation template Anda, `AWS::Lambda::Function` sumber daya menentukan fungsi Lambda Anda. Dalam sumber daya ini, atur properti berikut untuk membuat fungsi menggunakan arsip file.zip:

- `PackageType`- Setel ke `Zip`
- `Code`- Masukkan nama bucket Amazon S3 dan nama file.zip di dan bidang `S3Bucket` `S3Key`
- `Runtime`- Setel ke runtime yang Anda pilih

File.zip yang CloudFormation menghasilkan tidak boleh melebihi 4MB. Untuk mempelajari selengkapnya tentang penerapan fungsi menggunakan file.zip di CloudFormation, lihat AWS::Lambda: :Fungsi dalam Panduan Pengguna.CloudFormation

Terapkan Fungsi Acara Go Lambda dengan gambar kontainer

Ada dua cara untuk membangun image container untuk Go Lambda Event Function:

- Menggunakan sebuah AWS OS-only gambar dasar

  Go diimplementasikan secara berbeda dari runtime terkelola lainnya. Karena Go mengkompilasi secara native ke biner yang dapat dieksekusi, itu tidak memerlukan runtime bahasa khusus. Gunakan gambar OS-only dasar untuk membuat gambar Go untuk Lambda. Untuk membuat gambar kompatibel dengan Lambda, Anda harus menyertakan `aws-lambda-go/lambda` paket dalam gambar.

- Menggunakan non-AWS gambar dasar

  Anda dapat menggunakan gambar dasar alternatif dari registri kontainer lain, seperti Alpine Linux atau Debian. Anda juga dapat menggunakan gambar kustom yang dibuat oleh organisasi Anda. Untuk membuat gambar kompatibel dengan Lambda, Anda harus menyertakan `aws-lambda-go/lambda` paket dalam gambar.

> **Tip**
>
> Untuk mengurangi waktu yang dibutuhkan agar fungsi kontainer Lambda menjadi aktif, lihat Menggunakan build multi-tahap dalam dokumentasi Docker. Untuk membuat gambar kontainer yang efisien, ikuti Praktik terbaik untuk menulis Dockerfiles.

Halaman ini menjelaskan cara membuat, menguji, dan menyebarkan gambar kontainer untuk Lambda.

AWS gambar dasar untuk menerapkan fungsi Go

Go diimplementasikan secara berbeda dari runtime terkelola lainnya. Karena Go mengkompilasi secara native ke biner yang dapat dieksekusi, itu tidak memerlukan runtime bahasa khusus. Gunakan gambar OS-only dasar untuk menerapkan fungsi Go ke Lambda.

Nama	Pengidentifikasi	Sistem operasi	Tanggal pengusangan	Buat fungsi blok	Pembaruan fungsi blok
OS-only Runtime	`provided.al2023`	Amazon Linux 2023	30 Jun 2029	Jul 31, 2029	Agustus 31, 2029

Galeri Publik Registri Kontainer Elastis Amazon: gallery.ecr. aws/lambda/disediakan

Klien antarmuka Go runtime

Paket `aws-lambda-go/lambda` termasuk implementasi dari antarmuka runtime. Untuk contoh cara menggunakan `aws-lambda-go/lambda` dalam gambar Anda, lihat Menggunakan sebuah AWS OS-only gambar dasar atauMenggunakan non-AWS gambar dasar.

Menggunakan sebuah AWS OS-only gambar dasar

Go diimplementasikan secara berbeda dari runtime terkelola lainnya. Karena Go mengkompilasi secara native ke biner yang dapat dieksekusi, itu tidak memerlukan runtime bahasa khusus. Gunakan gambar OS-only dasar untuk membangun gambar kontainer untuk fungsi Go.

Tanda	Waktu berjalan	Sistem operasi	Dockerfile	penghentian
al2023	OS-only Runtime	Amazon Linux 2023	Dockerfile untuk OS-only Runtime aktif GitHub	30 Jun 2029

Untuk informasi selengkapnya tentang gambar dasar ini, lihat disediakan di galeri publik Amazon ECR.

Anda harus menyertakan go/lambda paket aws-lambda- dengan penangan Go Anda. Paket ini mengimplementasikan model pemrograman untuk Go, termasuk antarmuka runtime.

Prasyarat

Untuk menyelesaikan langkah-langkah di bagian ini, Anda harus memiliki hal-hal berikut:

- AWS CLI versi 2

- Docker (versi minimum 25.0.0)
- Plugin buildx Docker.
- Go

Membuat gambar dari gambar dasar.al2023 yang disediakan

Untuk membangun dan menerapkan fungsi Go dengan gambar dasar.al2023 yang **disediakan**

1. Buat direktori untuk proyek, dan kemudian beralih ke direktori itu.

```
mkdir hello
cd hello
```

2. Inisialisasi modul Go baru.

```
go mod init example.com/hello-world
```

3. Tambahkan pustaka lambda sebagai dependensi modul baru Anda.

```
go get github.com/aws/aws-lambda-go/lambda
```

4. Buat file bernama `main.go` dan kemudian buka di editor teks. Ini adalah kode untuk fungsi Lambda. Anda dapat menggunakan kode contoh berikut untuk pengujian, atau menggantinya dengan kode Anda sendiri.

```
package main

import (
 "context"
 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

func handler(ctx context.Context, event events.APIGatewayProxyRequest)
 (events.APIGatewayProxyResponse, error) {
 response := events.APIGatewayProxyResponse{
  StatusCode: 200,
  Body:       "\"Hello from Lambda!\"",
 }
 return response, nil
}
```

```
func main() {
 lambda.Start(handler)
}
```

5. Gunakan editor teks untuk membuat Dockerfile di direktori proyek Anda.

   - Contoh berikut Dockerfile menggunakan build multi-tahap di situs dokumentasi Docker. Dengan build multi-tahap, Anda dapat menggunakan gambar dasar yang berbeda di setiap langkah. Anda dapat menggunakan satu gambar, seperti image dasar Go di Docker Hub, untuk mengkompilasi kode Anda dan membangun biner yang dapat dieksekusi. Anda kemudian dapat menggunakan gambar yang berbeda, seperti`provided.al2023`, dalam `FROM` pernyataan akhir untuk menentukan gambar yang Anda terapkan ke Lambda. Build multi-tahap memisahkan proses build dari image penerapan akhir, sehingga gambar akhir hanya berisi file yang diperlukan untuk menjalankan aplikasi.
   - Anda dapat menggunakan `lambda.norpc` tag opsional untuk mengecualikan komponen Remote Procedure Call (RPC) dari pustaka lambda. Komponen RPC hanya diperlukan jika Anda menggunakan runtime Go 1.x yang tidak digunakan lagi. Mengecualikan RPC mengurangi ukuran paket penyebaran.
   - Perhatikan bahwa contoh Dockerfile tidak menyertakan instruksi USER. Saat Anda menyebarkan gambar kontainer ke Lambda, Lambda secara otomatis mendefinisikan pengguna Linux default dengan izin yang paling tidak memiliki hak istimewa. Ini berbeda dari perilaku Docker standar yang default ke `root` pengguna ketika tidak ada `USER` instruksi yang diberikan.

   Example— Multi-stage membangun Dockerfile

   > **Note**
   >
   > Pastikan bahwa versi Go yang Anda tentukan di Dockerfile Anda (misalnya,`golang:1.20`) adalah versi Go yang sama dengan yang Anda gunakan untuk membuat aplikasi Anda.

```
FROM golang:1.20 as build
WORKDIR /helloworld
# Copy dependencies list
COPY go.mod go.sum ./
```

```
# Build with optional lambda.norpc tag
COPY main.go .
RUN go build -tags lambda.norpc -o main main.go
# Copy artifacts to a clean image
FROM public.ecr.aws/lambda/provided:al2023
COPY --from=build /helloworld/main ./main
ENTRYPOINT [ "./main" ]
```

6. Buat image Docker dengan perintah docker build. Contoh berikut menamai gambar `docker-image` dan memberinya `test` tag. Untuk membuat gambar Anda kompatibel dengan Lambda, Anda harus menggunakan opsi. `--provenance=false`

```
docker buildx build --platform linux/amd64 --provenance=false -t docker-image:test
 .
```

> **Note**
>
> Perintah menentukan `--platform linux/amd64` opsi untuk memastikan bahwa container Anda kompatibel dengan lingkungan eksekusi Lambda terlepas dari arsitektur mesin build Anda. Jika Anda berniat untuk membuat fungsi Lambda menggunakan arsitektur set instruksi ARM64, pastikan untuk mengubah perintah untuk menggunakan opsi sebagai `--platform linux/arm64` gantinya.

(Opsional) Uji gambar secara lokal

Gunakan emulator antarmuka runtime untuk menguji gambar Anda secara lokal. Emulator antarmuka runtime disertakan dalam gambar `provided.al2023` dasar.

Untuk menjalankan emulator antarmuka runtime di mesin lokal Anda

1. Mulai gambar Docker dengan perintah docker run. Perhatikan hal-hal berikut:

   - `docker-image`adalah nama gambar dan `test` tag.
   - `./main`adalah `ENTRYPOINT` dari Dockerfile Anda.

```
docker run -d -p 9000:8080 \
--entrypoint /usr/local/bin/aws-lambda-rie \
docker-image:test ./main
```

Perintah ini menjalankan gambar sebagai wadah dan membuat titik akhir lokal di`localhost:9000/2015-03-31/functions/function/invocations`.

2. Dari jendela terminal baru, posting peristiwa ke titik akhir berikut menggunakan perintah curl:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
```

Perintah ini memanggil fungsi dengan peristiwa kosong dan mengembalikan respons. Beberapa fungsi mungkin memerlukan payload JSON. Contoh:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d
 '{"payload":"hello world!"}'
```

3. Dapatkan ID kontainer.

```
docker ps
```

4. Gunakan perintah docker kill untuk menghentikan kontainer. Dalam perintah ini, ganti `3766c4ab331c` dengan ID kontainer dari langkah sebelumnya.

```
docker kill 3766c4ab331c
```

Menyebarkan gambar

Untuk mengunggah gambar ke Amazon ECR dan membuat fungsi Lambda

1. Jalankan perintah get-login-password untuk mengautentikasi CLI Docker ke registri Amazon ECR Anda.
   - Tetapkan `--region` nilai ke Wilayah AWS tempat Anda ingin membuat repositori Amazon ECR.
   - Ganti `111122223333` dengan Akun AWS ID Anda.

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --
password-stdin 111122223333.dkr.ecr.us-east-1.amazonaws.com
```

2. Buat repositori di Amazon ECR menggunakan perintah create-repository.

```
aws ecr create-repository --repository-name hello-world --region us-east-1 --image-
scanning-configuration scanOnPush=true --image-tag-mutability MUTABLE
```

> **Note**
>
> Repositori Amazon ECR harus sama Wilayah AWS dengan fungsi Lambda.

Jika berhasil, Anda melihat respons seperti ini:

```
{
    "repository": {
        "repositoryArn": "arn:aws:ecr:us-east-1:111122223333:repository/hello-
world",
        "registryId": "111122223333",
        "repositoryName": "hello-world",
        "repositoryUri": "111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-
world",
        "createdAt": "2023-03-09T10:39:01+00:00",
        "imageTagMutability": "MUTABLE",
        "imageScanningConfiguration": {
            "scanOnPush": true
        },
        "encryptionConfiguration": {
            "encryptionType": "AES256"
        }
    }
}
```

3. Salin `repositoryUri` dari output pada langkah sebelumnya.
4. Jalankan perintah tag docker untuk menandai gambar lokal Anda ke repositori Amazon ECR Anda sebagai versi terbaru. Dalam perintah ini:

   - `docker-image:test`adalah nama dan tag gambar Docker Anda. Ini adalah nama gambar dan tag yang Anda tentukan dalam `docker build` perintah.
   - Ganti <ECRrepositoryUri> dengan `repositoryUri` yang Anda salin. Pastikan untuk menyertakan `:latest` di akhir URI.

```
docker tag docker-image:test <ECRrepositoryUri>:latest
```

Contoh:

```
docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

5. Jalankan perintah docker push untuk menyebarkan gambar lokal Anda ke repositori Amazon ECR. Pastikan untuk menyertakan `:latest` di akhir URI repositori.

```
docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

6. Buat peran eksekusi untuk fungsi tersebut, jika Anda belum memilikinya. Anda memerlukan Nama Sumber Daya Amazon (ARN) dari peran tersebut di langkah berikutnya.
7. Buat fungsi Lambda. Untuk`ImageUri`, tentukan URI repositori dari sebelumnya. Pastikan untuk menyertakan `:latest` di akhir URI.

```
aws lambda create-function \
  --function-name hello-world \
  --package-type Image \
  --code ImageUri=111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
  --role arn:aws:iam::111122223333:role/lambda-ex
```

> **Note**
>
> Anda dapat membuat fungsi menggunakan gambar di AWS akun yang berbeda, selama gambar berada di Wilayah yang sama dengan fungsi Lambda. Untuk informasi selengkapnya, lihat Izin lintas akun Amazon ECR.

8. Memanggil fungsi.

```
aws lambda invoke --function-name hello-world response.json
```

Anda akan melihat tanggapan seperti ini:

```
{
  "ExecutedVersion": "$LATEST",
```

```
  "StatusCode": 200
}
```

9. Untuk melihat output dari fungsi, periksa `response.json` file.

Untuk memperbarui kode fungsi, Anda harus membuat gambar lagi, mengunggah gambar baru ke repositori Amazon ECR, dan kemudian menggunakan perintah update-function-code untuk menyebarkan gambar ke fungsi Lambda.

Lambda menyelesaikan tag gambar ke intisari gambar tertentu. Ini berarti bahwa jika Anda mengarahkan tag gambar yang digunakan untuk menyebarkan fungsi ke gambar baru di Amazon ECR, Lambda tidak secara otomatis memperbarui fungsi untuk menggunakan gambar baru.

Untuk menyebarkan gambar baru ke fungsi Lambda yang sama, Anda harus menggunakan perintah update-function-code, meskipun tag gambar di Amazon ECR tetap sama. Dalam contoh berikut, `--publish` opsi membuat versi baru dari fungsi menggunakan gambar kontainer yang diperbarui.

```
aws lambda update-function-code \
  --function-name hello-world \
  --image-uri 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
  --publish
```

Menggunakan non-AWS gambar dasar

Anda dapat membuat gambar kontainer untuk Go dari gambar AWS non-dasar. Contoh Dockerfile dalam langkah-langkah berikut menggunakan gambar dasar Alpine.

Anda harus menyertakan go/lambda paket aws-lambda- dengan penangan Go Anda. Paket ini mengimplementasikan model pemrograman untuk Go, termasuk antarmuka runtime.

Prasyarat

Untuk menyelesaikan langkah-langkah di bagian ini, Anda harus memiliki hal-hal berikut:

- AWS CLI versi 2
- Docker (versi minimum 25.0.0)
- Plugin buildx Docker.
- Go

Membuat gambar dari gambar dasar alternatif

Untuk membangun dan menerapkan fungsi Go dengan image dasar Alpine

1. Buat direktori untuk proyek, dan kemudian beralih ke direktori itu.

```
mkdir hello
cd hello
```

2. Inisialisasi modul Go baru.

```
go mod init example.com/hello-world
```

3. Tambahkan pustaka lambda sebagai dependensi modul baru Anda.

```
go get github.com/aws/aws-lambda-go/lambda
```

4. Buat file bernama `main.go` dan kemudian buka di editor teks. Ini adalah kode untuk fungsi Lambda. Anda dapat menggunakan kode contoh berikut untuk pengujian, atau menggantinya dengan kode Anda sendiri.

```
package main

import (
 "context"
 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

func handler(ctx context.Context, event events.APIGatewayProxyRequest)
 (events.APIGatewayProxyResponse, error) {
 response := events.APIGatewayProxyResponse{
  StatusCode: 200,
  Body:       "\"Hello from Lambda!\"",
 }
 return response, nil
}

func main() {
 lambda.Start(handler)
}
```

5. Gunakan editor teks untuk membuat Dockerfile di direktori proyek Anda. Contoh berikut Dockerfile menggunakan gambar dasar Alpine. Perhatikan bahwa contoh Dockerfile tidak menyertakan instruksi USER. Saat Anda menyebarkan gambar kontainer ke Lambda, Lambda secara otomatis mendefinisikan pengguna Linux default dengan izin yang paling tidak memiliki hak istimewa. Ini berbeda dari perilaku Docker standar yang default ke `root` pengguna ketika tidak ada USER instruksi yang diberikan.

   Example Dockerfile

   > **Note**
   >
   > Pastikan bahwa versi Go yang Anda tentukan di Dockerfile Anda (misalnya,`golang:1.20`) adalah versi Go yang sama dengan yang Anda gunakan untuk membuat aplikasi Anda.

   ```
   FROM golang:1.20.2-alpine3.16 as build
   WORKDIR /helloworld
   # Copy dependencies list
   COPY go.mod go.sum ./
   # Build
   COPY main.go .
   RUN go build -o main main.go
   # Copy artifacts to a clean image
   FROM alpine:3.16
   COPY --from=build /helloworld/main /main
   ENTRYPOINT [ "/main" ]
   ```

6. Buat image Docker dengan perintah docker build. Contoh berikut menamai gambar `docker-image` dan memberinya `test` tag. Untuk membuat gambar Anda kompatibel dengan Lambda, Anda harus menggunakan opsi. `--provenance=false`

   ```
   docker buildx build --platform linux/amd64 --provenance=false -t docker-image:test
    .
   ```

   > **Note**
   >
   > Perintah menentukan `--platform linux/amd64` opsi untuk memastikan bahwa container Anda kompatibel dengan lingkungan eksekusi Lambda terlepas dari arsitektur

mesin build Anda. Jika Anda berniat untuk membuat fungsi Lambda menggunakan arsitektur set instruksi ARM64, pastikan untuk mengubah perintah untuk menggunakan opsi sebagai `--platform linux/arm64` gantinya.

(Opsional) Uji gambar secara lokal

Gunakan emulator antarmuka runtime untuk menguji gambar secara lokal. Anda dapat membangun emulator ke dalam gambar Anda atau menggunakan prosedur berikut untuk menginstalnya di mesin lokal Anda.

Untuk menginstal dan menjalankan emulator antarmuka runtime di mesin lokal Anda

1. Dari direktori proyek Anda, jalankan perintah berikut untuk mengunduh emulator antarmuka runtime (arsitektur x86-64) dari GitHub dan menginstalnya di mesin lokal Anda.

   Linux/macOS

   ```
   mkdir -p ~/.aws-lambda-rie && \
       curl -Lo ~/.aws-lambda-rie/aws-lambda-rie https://github.com/aws/aws-lambda-runtime-interface-emulator/releases/latest/download/aws-lambda-rie && \
       chmod +x ~/.aws-lambda-rie/aws-lambda-rie
   ```

   Untuk menginstal emulator arm64, ganti URL GitHub repositori di perintah sebelumnya dengan yang berikut:

   ```
   https://github.com/aws/aws-lambda-runtime-interface-emulator/releases/latest/download/aws-lambda-rie-arm64
   ```

   PowerShell

   ```
   $dirPath = "$HOME\.aws-lambda-rie"
   if (-not (Test-Path $dirPath)) {
       New-Item -Path $dirPath -ItemType Directory
   }

   $downloadLink = "https://github.com/aws/aws-lambda-runtime-interface-emulator/releases/latest/download/aws-lambda-rie"
   $destinationPath = "$HOME\.aws-lambda-rie\aws-lambda-rie"
   Invoke-WebRequest -Uri $downloadLink -OutFile $destinationPath
   ```

Untuk menginstal emulator arm64, ganti `$downloadLink` dengan yang berikut ini:

```
https://github.com/aws/aws-lambda-runtime-interface-emulator/releases/latest/
download/aws-lambda-rie-arm64
```

2. Mulai gambar Docker dengan perintah docker run. Perhatikan hal-hal berikut:

   - `docker-image`adalah nama gambar dan `test` tag.
   - `/main`adalah `ENTRYPOINT` dari Dockerfile Anda.

   Linux/macOS

   ```
   docker run --platform linux/amd64 -d -v ~/.aws-lambda-rie:/aws-lambda -p
    9000:8080 \
       --entrypoint /aws-lambda/aws-lambda-rie \
       docker-image:test \
           /main
   ```

   PowerShell

   ```
   docker run --platform linux/amd64 -d -v "$HOME\.aws-lambda-rie:/aws-lambda" -p
    9000:8080 `
   --entrypoint /aws-lambda/aws-lambda-rie `
   docker-image:test `
       /main
   ```

   Perintah ini menjalankan gambar sebagai wadah dan membuat titik akhir lokal di`localhost:9000/2015-03-31/functions/function/invocations`.

   > **Note**
   >
   > Jika Anda membuat image Docker untuk arsitektur set instruksi ARM64, pastikan untuk menggunakan `--platform linux/arm64` opsi sebagai gantinya. `--platform linux/amd64`

3. Posting acara ke titik akhir lokal.

Linux/macOS

Di Linux dan macOS, jalankan perintah berikut: `curl`

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
```

Perintah ini memanggil fungsi dengan peristiwa kosong dan mengembalikan respons. Jika Anda menggunakan kode fungsi Anda sendiri daripada kode fungsi sampel, Anda mungkin ingin memanggil fungsi dengan payload JSON. Contoh:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d
 '{"payload":"hello world!"}'
```

PowerShell

Dalam PowerShell, jalankan `Invoke-WebRequest` perintah berikut:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
invocations" -Method Post -Body '{}' -ContentType "application/json"
```

Perintah ini memanggil fungsi dengan peristiwa kosong dan mengembalikan respons. Jika Anda menggunakan kode fungsi Anda sendiri daripada kode fungsi sampel, Anda mungkin ingin memanggil fungsi dengan payload JSON. Contoh:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
invocations" -Method Post -Body '{"payload":"hello world!"}' -ContentType
 "application/json"
```

4. Dapatkan ID kontainer.

```
docker ps
```

5. Gunakan perintah docker kill untuk menghentikan kontainer. Dalam perintah ini, ganti `3766c4ab331c` dengan ID kontainer dari langkah sebelumnya.

```
docker kill 3766c4ab331c
```

Menyebarkan gambar

Untuk mengunggah gambar ke Amazon ECR dan membuat fungsi Lambda

1. Jalankan perintah get-login-password untuk mengautentikasi CLI Docker ke registri Amazon ECR Anda.

   - Tetapkan `--region` nilai ke Wilayah AWS tempat Anda ingin membuat repositori Amazon ECR.
   - Ganti `111122223333` dengan Akun AWS ID Anda.

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --
password-stdin 111122223333.dkr.ecr.us-east-1.amazonaws.com
```

2. Buat repositori di Amazon ECR menggunakan perintah create-repository.

```
aws ecr create-repository --repository-name hello-world --region us-east-1 --image-
scanning-configuration scanOnPush=true --image-tag-mutability MUTABLE
```

> **Note**
>
> Repositori Amazon ECR harus sama Wilayah AWS dengan fungsi Lambda.

Jika berhasil, Anda melihat respons seperti ini:

```
{
    "repository": {
        "repositoryArn": "arn:aws:ecr:us-east-1:111122223333:repository/hello-
world",
        "registryId": "111122223333",
        "repositoryName": "hello-world",
        "repositoryUri": "111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-
world",
        "createdAt": "2023-03-09T10:39:01+00:00",
        "imageTagMutability": "MUTABLE",
        "imageScanningConfiguration": {
            "scanOnPush": true
        },
        "encryptionConfiguration": {
```

```
            "encryptionType": "AES256"
        }
    }
}
```

3. Salin `repositoryUri` dari output pada langkah sebelumnya.
4. Jalankan perintah tag docker untuk menandai gambar lokal Anda ke repositori Amazon ECR Anda sebagai versi terbaru. Dalam perintah ini:
   - `docker-image:test`adalah nama dan tag gambar Docker Anda. Ini adalah nama gambar dan tag yang Anda tentukan dalam `docker build` perintah.
   - Ganti <ECRrepositoryUri> dengan `repositoryUri` yang Anda salin. Pastikan untuk menyertakan `:latest` di akhir URI.

```
docker tag docker-image:test <ECRrepositoryUri>:latest
```

   Contoh:

```
docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

5. Jalankan perintah docker push untuk menyebarkan gambar lokal Anda ke repositori Amazon ECR. Pastikan untuk menyertakan `:latest` di akhir URI repositori.

```
docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

6. Buat peran eksekusi untuk fungsi tersebut, jika Anda belum memilikinya. Anda memerlukan Nama Sumber Daya Amazon (ARN) dari peran tersebut di langkah berikutnya.
7. Buat fungsi Lambda. Untuk`ImageUri`, tentukan URI repositori dari sebelumnya. Pastikan untuk menyertakan `:latest` di akhir URI.

```
aws lambda create-function \
  --function-name hello-world \
  --package-type Image \
  --code ImageUri=111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
  --role arn:aws:iam::111122223333:role/lambda-ex
```

> **Note**
>
> Anda dapat membuat fungsi menggunakan gambar di AWS akun yang berbeda, selama gambar berada di Wilayah yang sama dengan fungsi Lambda. Untuk informasi selengkapnya, lihat Izin lintas akun Amazon ECR.

8. Memanggil fungsi.

```
aws lambda invoke --function-name hello-world response.json
```

Anda akan melihat tanggapan seperti ini:

```
{
  "ExecutedVersion": "$LATEST",
  "StatusCode": 200
}
```

9. Untuk melihat output dari fungsi, periksa `response.json` file.

Untuk memperbarui kode fungsi, Anda harus membuat gambar lagi, mengunggah gambar baru ke repositori Amazon ECR, dan kemudian menggunakan perintah update-function-code untuk menyebarkan gambar ke fungsi Lambda.

Lambda menyelesaikan tag gambar ke intisari gambar tertentu. Ini berarti bahwa jika Anda mengarahkan tag gambar yang digunakan untuk menyebarkan fungsi ke gambar baru di Amazon ECR, Lambda tidak secara otomatis memperbarui fungsi untuk menggunakan gambar baru.

Untuk menyebarkan gambar baru ke fungsi Lambda yang sama, Anda harus menggunakan perintah update-function-code, meskipun tag gambar di Amazon ECR tetap sama. Dalam contoh berikut, `--publish` opsi membuat versi baru dari fungsi menggunakan gambar kontainer yang diperbarui.

```
aws lambda update-function-code \
  --function-name hello-world \
  --image-uri 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
  --publish
```

Bekerja dengan lapisan untuk Fungsi Acara Go Lambda

Kami tidak menyarankan menggunakan lapisan untuk mengelola dependensi untuk fungsi Lambda yang ditulis dalam Go. Ini karena fungsi Lambda di Go dikompilasi menjadi satu executable, yang Anda berikan ke Lambda saat Anda menerapkan fungsi Anda. Executable ini berisi kode fungsi dikompilasi Anda, bersama dengan semua dependensinya. Menggunakan lapisan tidak hanya mempersulit proses ini, tetapi juga menyebabkan peningkatan waktu mulai dingin karena fungsi Anda perlu memuat rakitan ekstra secara manual ke dalam memori selama fase init.

Untuk menggunakan dependensi eksternal dengan penangan Go Anda, sertakan dependensi tersebut secara langsung dalam paket penerapan Anda. Dengan demikian, Anda menyederhanakan proses penerapan dan juga memanfaatkan pengoptimalan kompiler Go bawaan. Untuk contoh cara mengimpor dan menggunakan dependensi seperti AWS SDK for Go dalam fungsi Anda, lihat. the section called “Handler”

Log dan pantau Fungsi Acara Go Lambda

AWS Lambda secara otomatis memonitor Fungsi Peristiwa Lambda atas nama Anda dan mengirim log ke Amazon. CloudWatch Fungsi Anda dilengkapi dengan grup CloudWatch log Log dan aliran log untuk setiap instance fungsi Anda. Lingkungan runtime Lambda mengirimkan detail tentang setiap invokasi ke pengaliran log, dan menyampaikan log serta output lain dari kode fungsi Anda. Untuk informasi selengkapnya, lihat Mengirim log Fungsi Peristiwa Lambda ke Log CloudWatch.

Halaman ini menjelaskan cara menghasilkan keluaran log dari kode fungsi Lambda Anda, dan mengakses log menggunakan AWS Command Line Interface, konsol Lambda, atau konsol. CloudWatch

Bagian-bagian

- Membuat fungsi yang mengembalikan log
- Melihat log di konsol Lambda
- Melihat log di CloudWatch konsol
- Melihat log menggunakan AWS Command Line Interface (AWS CLI)
- Menghapus log

Membuat fungsi yang mengembalikan log

Untuk menghasilkan log dari kode fungsi, Anda dapat menggunakan metode di paket fmt, atau pustaka pencatatan apa pun yang menulis ke `stdout` atau `stderr`. Contoh berikut menggunakan paket log.

Example main.go — Penebangan

```
func handleRequest(ctx context.Context, event events.SQSEvent) (string, error) {
  // event
  eventJson, _ := json.MarshalIndent(event, "", "  ")
  log.Printf("EVENT: %s", eventJson)
  // environment variables
  log.Printf("REGION: %s", os.Getenv("AWS_REGION"))
  log.Println("ALL ENV VARS:")
  for _, element := range os.Environ() {
    log.Println(element)
  }
```

Example format log

```
START RequestId: dbda340c-xmpl-4031-8810-11bb609b4c71 Version: $LATEST
2020/03/27 03:40:05 EVENT: {
  "Records": [
    {
      "messageId": "19dd0b57-b21e-4ac1-bd88-01bbb068cb78",
      "receiptHandle": "MessageReceiptHandle",
      "body": "Hello from SQS!",
      "md5OfBody": "7b27xmplb47ff90a553787216d55d91d",
      "md5OfMessageAttributes": "",
      "attributes": {
        "ApproximateFirstReceiveTimestamp": "1523232000001",
        "ApproximateReceiveCount": "1",
        "SenderId": "123456789012",
        "SentTimestamp": "1523232000000"
      },
      ...
2020/03/27 03:40:05 AWS_LAMBDA_LOG_STREAM_NAME=2020/03/27/
[$LATEST]569cxmplc3c34c7489e6a97ad08b4419
2020/03/27 03:40:05 AWS_LAMBDA_FUNCTION_NAME=blank-go-function-9DV3XMPL6XBC
2020/03/27 03:40:05 AWS_LAMBDA_FUNCTION_MEMORY_SIZE=128
2020/03/27 03:40:05 AWS_LAMBDA_FUNCTION_VERSION=$LATEST
2020/03/27 03:40:05 AWS_EXECUTION_ENV=AWS_Lambda_go1.x
END RequestId: dbda340c-xmpl-4031-8810-11bb609b4c71
REPORT RequestId: dbda340c-xmpl-4031-8810-11bb609b4c71 Duration: 38.66 ms Billed
 Duration: 243 ms Memory Size: 128 MB Max Memory Used: 54 MB Init Duration: 203.69 ms
XRAY TraceId: 1-5e7d7595-212fxmpl9ee07c4884191322 SegmentId: 42ffxmpl0645f474 Sampled:
 true
```

Runtime Go akan mencatat baris `START`, `END`, dan `REPORT` untuk setiap invokasi. Baris laporan memberikan detail berikut.

Laporkan bidang data baris

- RequestId – ID permintaan unik untuk invokasi.
- Durasi – Jumlah waktu yang digunakan oleh metode handler fungsi Anda gunakan untuk memproses peristiwa.
- Durasi yang Ditagih – Jumlah waktu yang ditagihkan untuk invokasi.
- Ukuran Memori – Jumlah memori yang dialokasikan untuk fungsi.

- Memori Maks yang Digunakan – Jumlah memori yang digunakan oleh fungsi. Saat pemanggilan berbagi lingkungan eksekusi, Lambda melaporkan memori maksimum yang digunakan di semua pemanggilan. Perilaku ini dapat menghasilkan nilai yang dilaporkan lebih tinggi dari yang diharapkan.
- Durasi Init – Untuk permintaan pertama yang dilayani, lama waktu yang diperlukan runtime untuk memuat fungsi dan menjalankan kode di luar metode handler.
- XRAY TraceId — Untuk permintaan yang dilacak, ID AWS X-Ray jejak.
- SegmentId— Untuk permintaan yang dilacak, ID X-Ray segmen.
- Diambil Sampel – Untuk permintaan yang dilacak, hasil pengambilan sampel.

Melihat log di konsol Lambda

Anda dapat menggunakan konsol Lambda untuk melihat keluaran log setelah Anda menjalankan Fungsi Peristiwa Lambda.

Jika kode Anda dapat diuji dari editor Kode tertanam, Anda menemukan log dalam hasil eksekusi. Saat Anda menggunakan fitur pengujian konsol untuk menjalankan fungsi, Anda menemukan Keluaran Log di bagian Detail.

Melihat log di CloudWatch konsol

Anda dapat menggunakan CloudWatch konsol Amazon untuk melihat log untuk semua pemanggilan Fungsi Peristiwa Lambda.

Untuk melihat log di CloudWatch konsol

1. Buka halaman Grup log di CloudWatch konsol.
2. Pilih grup log untuk fungsi Anda (/aws/lambda/***your-function-name***).
3. Pilih pengaliran log.

Setiap aliran log sesuai dengan instans fungsi Anda. Aliran log muncul saat Anda memperbarui fungsi Lambda, dan saat instance tambahan dibuat untuk menangani pemanggilan bersamaan. Untuk menemukan log untuk pemanggilan tertentu, kami sarankan untuk menginstrumentasi fungsi Anda dengan. AWS X-Ray X-Ray mencatat detail tentang permintaan dan aliran log dalam jejak.

Melihat log menggunakan AWS Command Line Interface (AWS CLI)

AWS CLI Ini adalah alat sumber terbuka yang dapat Anda gunakan untuk berinteraksi dengan AWS layanan menggunakan perintah di shell baris perintah Anda. Untuk menyelesaikan langkah-langkah di bagian ini, Anda harus memiliki AWS CLI versi 2.

Anda dapat menggunakan AWS CLI untuk mengambil log untuk invokasi menggunakan opsi perintah `--log-type`. Respons berisi bidang `LogResult` yang memuat hingga 4 KB log berkode base64 dari invokasi.

Example mengambil ID log

Contoh berikut menunjukkan cara mengambil ID log dari `LogResult` untuk fungsi bernama `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

Anda akan melihat output berikut:

```
{
    "StatusCode": 200,
    "LogResult":
 "U1RBUlQgUmVxdWVzdElkOiA4N2QwNDRiOC1mMTU0LTExZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "ExecutedVersion": "$LATEST"
}
```

Example mendekode log

Pada prompt perintah yang sama, gunakan utilitas `base64` untuk mendekodekan log. Contoh berikut menunjukkan cara mengambil log berkode base64 untuk `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail \
--query 'LogResult' --output text --cli-binary-format raw-in-base64-out | base64 --
decode
```

cli-binary-formatOpsi ini diperlukan jika Anda menggunakan AWS CLI versi 2. Untuk menjadikan ini pengaturan default, jalankan`aws configure set cli-binary-format raw-in-base64-out`. Untuk informasi selengkapnya, lihat opsi baris perintah global yang AWS CLI didukung di Panduan AWS Command Line Interface Pengguna untuk Versi 2.

Anda akan melihat output berikut:

```
START RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Version: $LATEST
"AWS_SESSION_TOKEN": "AgoJb3JpZ2luX2VjELj...", "_X_AMZN_TRACE_ID": "Root=1-5d02e5ca-
f5792818b6fe8368e5b51d50;Parent=191db58857df8395;Sampled=0"",ask/lib:/opt/lib",
END RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8
REPORT RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8  Duration: 79.67 ms      Billed
 Duration: 80 ms         Memory Size: 128 MB     Max Memory Used: 73 MB
```

Utilitas `base64` tersedia di Linux, macOS, dan Ubuntu pada Windows. Pengguna macOS mungkin harus menggunakan `base64 -D`.

Example Skrip get-logs.sh

Pada prompt perintah yang sama, gunakan script berikut untuk mengunduh lima peristiwa log terakhir. Skrip menggunakan `sed` untuk menghapus kutipan dari file output, dan akan tidur selama 15 detik untuk memberikan waktu agar log tersedia. Output mencakup respons dari Lambda dan output dari perintah `get-log-events`.

Salin konten dari contoh kode berikut dan simpan dalam direktori proyek Lambda Anda sebagai `get-logs.sh`.

cli-binary-formatOpsi ini diperlukan jika Anda menggunakan AWS CLI versi 2. Untuk menjadikan ini pengaturan default, jalankan`aws configure set cli-binary-format raw-in-base64-out`. Untuk informasi selengkapnya, lihat opsi baris perintah global yang AWS CLI didukung di Panduan AWS Command Line Interface Pengguna untuk Versi 2.

```
#!/bin/bash
aws lambda invoke --function-name my-function --cli-binary-format raw-in-base64-out --
payload '{"key": "value"}' out
sed -i'' -e 's/"//g' out
sleep 15
aws logs get-log-events --log-group-name /aws/lambda/my-function --log-stream-
name stream1 --limit 5
```

Example macOS dan Linux (khusus)

Dalam prompt perintah yang sama, pengguna macOS dan Linux mungkin perlu menjalankan perintah berikut untuk memastikan skrip dapat dieksekusi.

```
chmod -R 755 get-logs.sh
```

Example mengambil lima log acara terakhir

Pada prompt perintah yang sama, gunakan skrip berikut untuk mendapatkan lima log acara terakhir.

```
./get-logs.sh
```

Anda akan melihat output berikut:

```
{
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}
{
    "events": [
        {
            "timestamp": 1559763003171,
            "message": "START RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf Version:
 $LATEST\n",
            "ingestionTime": 1559763003309
        },
        {
            "timestamp": 1559763003173,
            "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tENVIRONMENT VARIABLES\r{\r  \"AWS_LAMBDA_FUNCTION_VERSION\": \"$LATEST\",
\r ...",
            "ingestionTime": 1559763018353
        },
        {
            "timestamp": 1559763003173,
            "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tEVENT\r{\r  \"key\": \"value\"\r}\n",
            "ingestionTime": 1559763018353
        },
        {
            "timestamp": 1559763003218,
            "message": "END RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf\n",
            "ingestionTime": 1559763018353
        },
        {
            "timestamp": 1559763003218,
            "message": "REPORT RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tDuration: 26.73 ms\tBilled Duration: 27 ms \tMemory Size: 128 MB\tMax Memory Used: 75
 MB\t\n",
```

```
            "ingestionTime": 1559763018353
        }
    ],
    "nextForwardToken": "f/34783877304859518393868359594929986069206639495374241795",
    "nextBackwardToken": "b/34783877303811383369537420289090800615709599058929582080"
}
```

Menghapus log

Grup log tidak terhapus secara otomatis ketika Anda menghapus suatu fungsi. Untuk menghindari penyimpanan log secara tidak terbatas, hapus kelompok log, atau lakukan konfigurasi periode penyimpanan, yang setelahnya log akan dihapus secara otomatis.

Menginstrumentasi kode Go di AWS Lambda

Lambda terintegrasi dengan AWS X-Ray untuk membantu Anda melacak, men-debug, dan mengoptimalkan aplikasi Lambda. Anda dapat menggunakan X-Ray untuk melacak permintaan saat melintasi sumber daya dalam aplikasi Anda, yang mungkin termasuk fungsi Lambda dan layanan lainnya. AWS

Untuk mengirim data penelusuran X-Ray, Anda dapat menggunakan salah satu dari dua pustaka SDK:

- AWS Distro for OpenTelemetry (ADOT) — Distribusi SDK (OTel) yang aman, siap produksi, dan AWS didukung. OpenTelemetry
- AWS X-Ray SDK for Go — SDK untuk menghasilkan dan mengirim data jejak ke. X-Ray

Setiap SDK menawarkan cara untuk mengirim data telemetri Anda ke layanan. X-Ray Anda kemudian dapat menggunakan X-Ray untuk melihat, memfilter, dan mendapatkan wawasan tentang metrik kinerja aplikasi Anda untuk mengidentifikasi masalah dan peluang untuk pengoptimalan.

> **Important**
>
> The X-Ray dan Powertools untuk AWS Lambda SDK adalah bagian dari solusi instrumentasi terintegrasi ketat yang ditawarkan oleh. AWS Lapisan Lambda ADOT adalah bagian dari standar industri untuk melacak instrumentasi yang mengumpulkan lebih banyak data secara umum, tetapi mungkin tidak cocok untuk semua kasus penggunaan. Anda dapat menerapkan penelusuran ujung ke ujung dalam X-Ray menggunakan salah satu solusi. Untuk mempelajari lebih lanjut tentang memilih di antara keduanya, lihat Memilih antara AWS Distro untuk Telemetri Terbuka dan SDK. X-Ray

Bagian-bagian

- Menggunakan ADOT untuk instrumen fungsi Go Anda
- Menggunakan X-Ray SDK untuk instrumen fungsi Go Anda
- Mengaktifkan penelusuran dengan konsol Lambda
- Mengaktifkan penelusuran dengan Lambda API
- Mengaktifkan penelusuran dengan CloudFormation
- Menafsirkan jejak X-Ray

Menggunakan ADOT untuk instrumen fungsi Go Anda

ADOT menyediakan lapisan Lambda yang dikelola sepenuhnya yang mengemas semua yang Anda butuhkan untuk mengumpulkan data telemetri menggunakan OTel SDK. Dengan menggunakan lapisan ini, Anda dapat menginstruksikan fungsi Lambda Anda tanpa harus memodifikasi kode fungsi apa pun. Anda juga dapat mengonfigurasi lapisan Anda untuk melakukan inisialisasi khusus OTel. Untuk informasi selengkapnya, lihat Konfigurasi khusus untuk Kolektor ADOT di Lambda dalam dokumentasi ADOT.

Untuk runtime Go, Anda dapat menambahkan layer Lambda AWS terkelola untuk ADOT Go untuk secara otomatis menginstruksikan fungsi Anda. Untuk petunjuk rinci tentang cara menambahkan layer ini, lihat AWS Distro untuk OpenTelemetry Lambda Support for Go di dokumentasi ADOT. Untuk petunjuk tentang mengonversi peristiwa OpenTelemetry API Telemetri Lambda ke rentang, lihat. the section called “Mengonversi acara ke OTel Spans”

Menggunakan X-Ray SDK untuk instrumen fungsi Go Anda

Untuk merekam detail tentang panggilan yang dilakukan fungsi Lambda ke sumber daya lain di aplikasi, Anda juga dapat menggunakan AWS X-Ray SDK for Go. Untuk mendapatkan SDK, unduh SDK dari GitHub repositorinya dengan: `go get`

```
go get github.com/aws/aws-xray-sdk-go
```

Untuk instrumen klien AWS SDK, teruskan klien ke `xray.AWS()` metode. Anda kemudian dapat melacak panggilan dengan menggunakan `WithContext` versi metode.

```
svc := s3.New(session.New())
xray.AWS(svc.Client)
...
svc.ListBucketsWithContext(ctx aws.Context, input *ListBucketsInput)
```

Setelah Anda menambahkan dependensi yang benar dan membuat perubahan kode yang diperlukan, aktifkan penelusuran dalam konfigurasi fungsi Anda melalui konsol Lambda atau API.

Mengaktifkan penelusuran dengan konsol Lambda

Untuk mengaktifkan penelusuran aktif di Fungsi Peristiwa Lambda Anda dengan konsol, ikuti langkah-langkah berikut:

Untuk mengaktifkan penelusuran aktif

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih Fungsi Acara.
3. Pilih Konfigurasi dan kemudian pilih Alat Pemantauan dan operasi.
4. Di bawah Alat pemantauan tambahan, pilih Edit.
5. Di bawah Sinyal CloudWatch Aplikasi dan AWS X-Ray, pilih Aktifkan untuk jejak layanan Lambda.
6. Pilih Simpan.

Mengaktifkan penelusuran dengan Lambda API

Konfigurasikan penelusuran pada fungsi Lambda Anda dengan AWS CLI AWS atau SDK, gunakan operasi API berikut:

- UpdateFunctionConfiguration
- GetFunctionConfiguration
- CreateFunction

Contoh AWS CLI perintah berikut memungkinkan penelusuran aktif pada fungsi bernama my-function.

```
aws lambda update-function-configuration --function-name my-function \
--tracing-config Mode=Active
```

Mode penelusuran adalah bagian dari konfigurasi khusus versi saat Anda memublikasikan versi fungsi Anda. Anda tidak dapat mengubah mode pelacakan pada versi yang telah diterbitkan.

Mengaktifkan penelusuran dengan CloudFormation

Untuk mengaktifkan penelusuran pada `AWS::Lambda::Function` sumber daya dalam CloudFormation templat, gunakan `TracingConfig` properti.

Example fungsi-inline.yml. — Konfigurasi penelusuran

```
Resources:
```

```
  function:
    Type: AWS::Lambda::Function
    Properties:
      TracingConfig:
        Mode: Active
      ...
```

Untuk sumber `AWS::Serverless::Function` daya AWS Serverless Application Model (AWS SAM), gunakan `Tracing` properti.

Example template.yml — Konfigurasi penelusuran

```
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      Tracing: Active
      ...
```

Menafsirkan jejak X-Ray

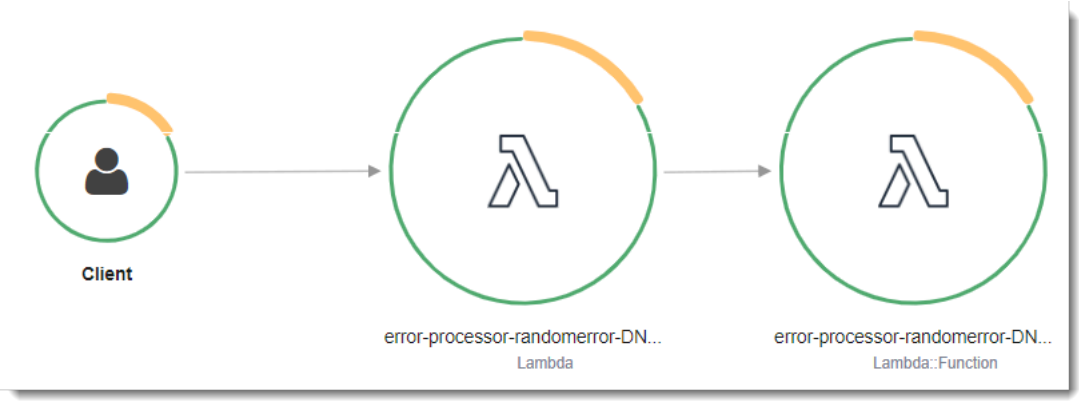
Fungsi Acara Anda memerlukan izin untuk mengunggah data jejak ke X-Ray. Saat Anda mengaktifkan penelusuran di konsol Lambda, Lambda menambahkan izin yang diperlukan ke peran eksekusi Fungsi Acara Anda. Atau, tambahkan kebijakan AWSXRayDaemonWriteAccess ke peran eksekusi.

Setelah mengonfigurasi penelusuran aktif, Anda dapat mengamati permintaan tertentu melalui aplikasi Anda. Grafik X-Ray layanan menunjukkan informasi tentang aplikasi Anda dan semua komponennya. Contoh berikut menunjukkan aplikasi dengan dua fungsi. Fungsi utama memproses kejadian dan terkadang mengembalikan kesalahan. Fungsi kedua di bagian atas memproses kesalahan yang muncul di grup log pertama dan menggunakan AWS SDK untuk memanggil X-Ray, Amazon Simple Storage Service (Amazon S3) Service, dan Amazon Logs. CloudWatch

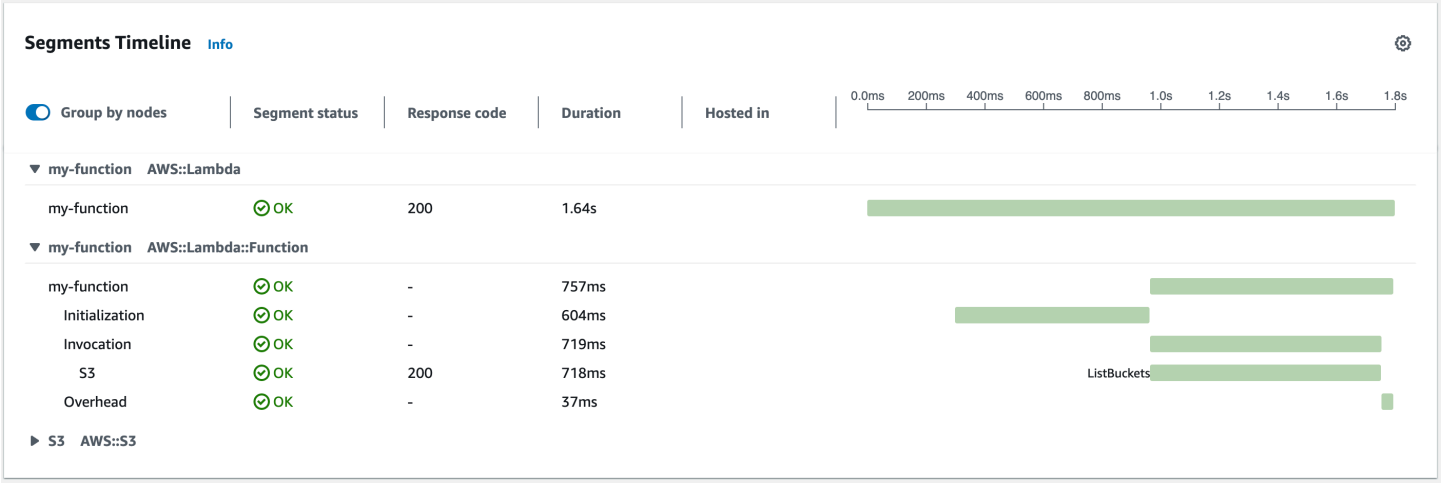


X-Ray tidak melacak semua permintaan ke aplikasi Anda. X-Ray menerapkan algoritma pengambilan sampel untuk memastikan bahwa penelusuran efisien, sambil tetap memberikan sampel yang representatif dari semua permintaan. Tingkat pengambilan sampel adalah 1 permintaan per detik dan 5 persen dari permintaan tambahan. Anda tidak dapat mengonfigurasi laju X-Ray pengambilan sampel untuk fungsi Anda.

Di X-Ray, jejak mencatat informasi tentang permintaan yang diproses oleh satu atau beberapa layanan. Lambda mencatat 2 segmen per jejak, yang menciptakan dua node pada grafik layanan. Gambar berikut menyoroti dua node ini:



Node pertama di sebelah kiri mewakili layanan Lambda, yang menerima permintaan pemanggilan. Node kedua mewakili fungsi Lambda spesifik Anda. Contoh berikut menunjukkan jejak dengan dua segmen ini. Keduanya bernama fungsi saya, tetapi yang satu memiliki asal `AWS::Lambda` dan yang lainnya memiliki asal usul. `AWS::Lambda::Function` Jika `AWS::Lambda` segmen menunjukkan kesalahan, layanan Lambda mengalami masalah. Jika `AWS::Lambda::Function` segmen menunjukkan kesalahan, fungsi Anda mengalami masalah.



Contoh ini memperluas `AWS::Lambda::Function` segmen untuk menunjukkan tiga subsegmennya.

> **Note**
>
> AWS saat ini menerapkan perubahan pada layanan Lambda. Karena perubahan ini, Anda mungkin melihat perbedaan kecil antara struktur dan konten pesan log sistem dan segmen jejak yang dipancarkan oleh Fungsi Peristiwa Lambda yang berbeda di Anda. Akun AWS Contoh jejak yang ditunjukkan di sini menggambarkan segmen fungsi gaya lama. Perbedaan antara segmen gaya lama dan baru dijelaskan dalam paragraf berikut.
> Perubahan ini akan diterapkan selama beberapa minggu mendatang, dan semua fungsi di semua Wilayah AWS kecuali China GovCloud dan wilayah akan bertransisi untuk menggunakan pesan log format baru dan segmen pelacakan.

Segmen fungsi gaya lama berisi subsegmen berikut:

- Inisialisasi – Mewakili waktu yang dihabiskan untuk memuat fungsi dan menjalankan kode inisialisasi. Subsegmen ini hanya muncul untuk peristiwa pertama yang diproses oleh setiap instance fungsi Anda.
- Doa - Merupakan waktu yang dihabiskan untuk menjalankan kode handler Anda.
- Overhead - Merupakan waktu yang dihabiskan runtime Lambda untuk mempersiapkan diri untuk menangani acara berikutnya.

Segmen fungsi gaya baru tidak berisi subsegmen. `Invocation` Sebagai gantinya, subsegmen pelanggan dilampirkan langsung ke segmen fungsi. Untuk informasi lebih lanjut tentang struktur segmen fungsi gaya lama dan baru, lihat. the section called “Memahami X-Ray jejak”

Anda juga dapat melakukan instrumentasi klien HTTP, merekam kueri SQL, dan membuat subsegmen khusus dengan anotasi dan metadata. Untuk informasi selengkapnya, lihat AWS X-Ray SDK for Go di Panduan AWS X-Ray Pengembang.

> **Harga**
>
> Anda dapat menggunakan X-Ray tracing gratis setiap bulan hingga batas tertentu sebagai bagian dari Tingkat AWS Gratis. Di luar ambang batas itu, X-Ray biaya untuk penyimpanan dan pengambilan jejak. Untuk informasi lebih lanjut, lihat Harga AWS X-Ray.

Membangun Fungsi Acara Lambda dengan C

Anda dapat menjalankan aplikasi.NET Anda sebagai Fungsi Peristiwa Lambda menggunakan runtime .NET 8 yang dikelola, runtime kustom, atau gambar kontainer. Setelah kode aplikasi dikompilasi, Anda dapat menerapkannya ke Lambda baik sebagai file.zip atau gambar kontainer. Lambda menyediakan runtime berikut untuk bahasa.NET:

Untuk informasi tentang menggunakan lapisan.NET, lihatthe section called “Lapisan”. Untuk panduan tentang pengujian fungsi C# Anda, lihatthe section called “Pengujian”.

Nama	Pengidentifikasi	Sistem operasi	Tanggal pengusangan	Buat fungsi blok	Pembaruan fungsi blok
.NET 10	`dotnet10`	Amazon Linux 2023	Nov 14, 2028	14 Des 2028	15 Jan 2029
.NET 9 (hanya wadah)	`dotnet9`	Amazon Linux 2023	Nov 10, 2026	Tidak dijadwalkan	Tidak dijadwalkan
.NET 8	`dotnet8`	Amazon Linux 2023	Nov 10, 2026	Jul 29, 2027	Agustus 31, 2027

Menyiapkan lingkungan pengembangan .NET Anda

Untuk mengembangkan dan membangun Fungsi Acara Lambda, Anda dapat menggunakan salah satu lingkungan pengembangan terintegrasi (IDE) .NET yang tersedia secara umum, termasuk Microsoft Visual Studio, Visual Studio Code, dan JetBrains Rider. Untuk menyederhanakan pengalaman pengembangan Anda, AWS sediakan satu set template proyek.NET, serta antarmuka baris `Amazon.Lambda.Tools` perintah (CLI).

Jalankan perintah.NET CLI berikut untuk menginstal template proyek dan alat baris perintah ini.

Menginstal template proyek.NET

Untuk menginstal template proyek, jalankan perintah berikut:

```
dotnet new install Amazon.Lambda.Templates
```

Menginstal dan memperbarui alat CLI

Jalankan perintah berikut untuk menginstal, memperbarui, dan menghapus `Amazon.Lambda.Tools` CLI.

Untuk menginstal alat baris perintah:

```
dotnet tool install -g Amazon.Lambda.Tools
```

Untuk memperbarui alat baris perintah:

```
dotnet tool update -g Amazon.Lambda.Tools
```

Untuk menghapus alat baris perintah:

```
dotnet tool uninstall -g Amazon.Lambda.Tools
```

Tentukan penangan Fungsi Peristiwa Lambda di C

Lambda Event Function handler adalah metode dalam kode fungsi Anda yang memproses peristiwa. Saat fungsi Anda diaktifkan, Lambda menjalankan metode handler. Fungsi Anda berjalan sampai handler mengembalikan respons, keluar, atau waktu habis.

Halaman ini menjelaskan cara bekerja dengan penangan Fungsi Peristiwa Lambda di C# untuk bekerja dengan runtime yang dikelola .NET, termasuk opsi untuk penyiapan proyek, konvensi penamaan, dan praktik terbaik. Halaman ini juga menyertakan contoh fungsi C# Lambda yang mengambil informasi tentang pesanan, menghasilkan tanda terima file teks, dan menempatkan file ini di bucket Amazon Simple Storage Service (S3). Untuk informasi tentang cara menerapkan fungsi Anda setelah menulisnya, lihat the section called “Paket deployment” atauthe section called “Deploy gambar kontainer”.

Topik

- Menyiapkan proyek handler C# Anda
- Contoh C # Lambda Kode Fungsi Acara
- Penangan perpustakaan kelas
- Penangan perakitan yang dapat dieksekusi
- Tanda tangan handler yang valid untuk fungsi C #
- Konvensi penamaan handler
- Serialisasi dalam Fungsi Acara C# Lambda
- File-based fungsi
- Mengakses dan menggunakan objek konteks Lambda
- Menggunakan SDK untuk .NET v3 di handler Anda
- Mengakses variabel lingkungan
- Menggunakan status global
- Sederhanakan kode fungsi dengan kerangka kerja Anotasi Lambda
- Praktik terbaik kode untuk Fungsi Acara C# Lambda

Menyiapkan proyek handler C# Anda

Saat bekerja dengan Lambda Event Functions di C #, prosesnya melibatkan penulisan kode Anda, lalu menerapkan kode Anda ke Lambda. Ada dua model eksekusi yang berbeda untuk menerapkan

fungsi Lambda di .NET: pendekatan perpustakaan kelas dan pendekatan perakitan yang dapat dieksekusi.

Dalam pendekatan pustaka kelas, Anda mengemas kode fungsi Anda sebagai .NET assembly (`.dll`) dan menerapkannya ke Lambda dengan runtime `dotnet8` terkelola .NET (). Untuk nama handler, Lambda mengharapkan string dalam format. `AssemblyName::Namespace.Classname::Methodname` Selama fase inisialisasi fungsi, kelas fungsi Anda diinisialisasi, dan kode apa pun dalam konstruktor dijalankan.

Dalam pendekatan perakitan yang dapat dieksekusi, Anda menggunakan fitur pernyataan tingkat atas yang pertama kali diperkenalkan di C # 9. Pendekatan ini menghasilkan perakitan yang dapat dieksekusi yang dijalankan Lambda setiap kali menerima perintah pemanggilan untuk fungsi Anda. Dalam pendekatan ini, Anda juga menggunakan .NET managed runtime (`dotnet8`). Untuk nama handler, Anda memberi Lambda nama rakitan yang dapat dieksekusi untuk dijalankan.

Contoh utama pada halaman ini menggambarkan pendekatan perpustakaan kelas. Anda dapat menginisialisasi proyek C# Lambda Anda dengan berbagai cara, tetapi cara termudah adalah dengan menggunakan CLI .NET dengan CLI. `Amazon.Lambda.Tools` Siapkan `Amazon.Lambda.Tools` CLI dengan mengikuti langkah-langkah di. the section called “Lingkungan pengembangan” Kemudian, inisialisasi proyek Anda dengan perintah berikut:

```
dotnet new lambda.EmptyFunction --name ExampleCS
```

Perintah ini menghasilkan struktur file berikut:

```
/project-root
    # src
        # ExampleCS
            # Function.cs (contains main handler)
            # Readme.md
            # aws-lambda-tools-defaults.json
            # ExampleCS.csproj
    # test
         # ExampleCS.Tests
            # FunctionTest.cs (contains main handler)
            # ExampleCS.Tests.csproj
```

Dalam struktur file ini, logika handler utama untuk fungsi Anda berada di `Function.cs` file.

Contoh C # Lambda Kode Fungsi Acara

Contoh kode fungsi C# Lambda berikut mengambil informasi tentang pesanan, menghasilkan tanda terima file teks, dan menempatkan file ini di bucket Amazon S3.

Example **`Function.cs`**Fungsi Acara Lambda

```
using System;
using System.Text;
using System.Threading.Tasks;
using Amazon.Lambda.Core;
using Amazon.S3;
using Amazon.S3.Model;

// Assembly attribute to enable Lambda function logging
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer

namespace ExampleLambda;

public class Order
{
    public string OrderId { get; set; } = string.Empty;
    public double Amount { get; set; }
    public string Item { get; set; } = string.Empty;
}

public class OrderHandler
{
    private static readonly AmazonS3Client s3Client = new();

    public async Task<string> HandleRequest(Order order, ILambdaContext context)
    {
        try
        {
            string? bucketName = Environment.GetEnvironmentVariable("RECEIPT_BUCKET");
            if (string.IsNullOrWhiteSpace(bucketName))
            {
                throw new ArgumentException("RECEIPT_BUCKET environment variable is not
 set");
            }

            string receiptContent = $"OrderID: {order.OrderId}\nAmount:
 ${order.Amount:F2}\nItem: {order.Item}";
```

```
            string key = $"receipts/{order.OrderId}.txt";

            await UploadReceiptToS3(bucketName, key, receiptContent);

            context.Logger.LogInformation($"Successfully processed order
 {order.OrderId} and stored receipt in S3 bucket {bucketName}");
            return "Success";
        }
        catch (Exception ex)
        {
            context.Logger.LogError($"Failed to process order: {ex.Message}");
            throw;
        }
    }

    private async Task UploadReceiptToS3(string bucketName, string key, string
 receiptContent)
    {
        try
        {
            var putRequest = new PutObjectRequest
            {
                BucketName = bucketName,
                Key = key,
                ContentBody = receiptContent,
                ContentType = "text/plain"
            };

            await s3Client.PutObjectAsync(putRequest);
        }
        catch (AmazonS3Exception ex)
        {
            throw new Exception($"Failed to upload receipt to S3: {ex.Message}", ex);
        }
    }
}
```

`Function.cs`File ini berisi bagian kode berikut:

- `using`pernyataan: Gunakan ini untuk mengimpor kelas C# yang diperlukan oleh fungsi Lambda Anda.

- `[assembly: LambdaSerializer(...)]`: `LambdaSerializer` adalah atribut assembly yang memberi tahu Lambda untuk secara otomatis mengonversi muatan acara JSON menjadi objek C # sebelum meneruskannya ke fungsi Anda.
- `namespace ExampleLambda`: Ini mendefinisikan namespace. Di C #, nama namespace tidak harus cocok dengan nama file.
- `public class Order {...}`: Ini mendefinisikan bentuk peristiwa input yang diharapkan.
- `public class OrderHandler {...}`: Ini mendefinisikan kelas C # Anda. Di dalamnya, Anda akan mendefinisikan metode handler utama dan metode pembantu lainnya.
- `private static readonly AmazonS3Client s3Client = new();`: Ini menginisialisasi klien Amazon S3 dengan rantai penyedia kredensi default, di luar metode penangan utama. Hal ini menyebabkan Lambda menjalankan kode ini selama fase inisialisasi.
- `public async ... HandleRequest (Order order, ILambdaContext context)`: Ini adalah metode handler utama, yang berisi logika aplikasi utama Anda.
- `private async Task UploadReceiptToS3(...) {}`: Ini adalah metode helper yang direferensikan oleh metode `handleRequest` handler utama.

Karena fungsi ini memerlukan klien SDK Amazon S3, Anda harus menambahkannya ke dependensi proyek Anda. Anda dapat melakukannya dengan menavigasi ke `src/ExampleCS` dan menjalankan perintah berikut:

```
dotnet add package AWSSDK.S3
```

Tambahkan informasi metadata ke aws-lambda-tools-defaults.json

Secara default, `aws-lambda-tools-defaults.json` file yang dihasilkan tidak berisi `profile` atau `region` informasi untuk fungsi Anda. Selain itu, perbarui `function-handler` string ke nilai yang benar (`ExampleCS::ExampleLambda.OrderHandler::HandleRequest`). Anda dapat membuat pembaruan ini secara manual dan menambahkan metadata yang diperlukan untuk menggunakan profil dan wilayah kredensil tertentu untuk fungsi Anda. Misalnya, `aws-lambda-tools-defaults.json` file Anda akan terlihat mirip dengan ini:

```
{
  "Information": [
    "This file provides default values for the deployment wizard inside Visual Studio
 and the AWS Lambda commands added to the .NET Core CLI.",
    "To learn more about the Lambda commands with the .NET Core CLI execute the
 following command at the command line in the project root directory.",
```

```
    "dotnet lambda help",
    "All the command line options for the Lambda command can be specified in this
 file."
  ],
  "profile": "default",
  "region": "us-east-1",
  "configuration": "Release",
  "function-architecture": "x86_64",
  "function-runtime": "dotnet8",
  "function-memory-size": 512,
  "function-timeout": 30,
  "function-handler": "ExampleCS::ExampleLambda.OrderHandler::HandleRequest"
}
```

Agar fungsi ini berfungsi dengan baik, peran pelaksanaannya harus memungkinkan `s3:PutObject` tindakan. Juga, pastikan bahwa Anda mendefinisikan variabel `RECEIPT_BUCKET` lingkungan. Setelah pemanggilan berhasil, bucket Amazon S3 harus berisi file tanda terima.

Penangan perpustakaan kelas

Contoh kode utama pada halaman ini menggambarkan handler perpustakaan kelas. Penangan perpustakaan kelas memiliki struktur sebagai berikut:

```
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer

namespace NAMESPACE;

...

public class CLASSNAME {
    public async Task<string> METHODNAME (...) {
    ...
    }
}
```

Saat Anda membuat Fungsi Peristiwa Lambda, Anda perlu memberikan Lambda informasi tentang penangan fungsi Anda dalam bentuk string di bidang Handler. Ini memberi tahu Lambda metode mana dalam kode Anda untuk dijalankan ketika fungsi Anda dipanggil. Di C #, untuk penangan perpustakaan kelas, format string handler adalah`ASSEMBLY::TYPE::METHOD`, di mana:

- `ASSEMBLY`adalah nama file perakitan .NET untuk aplikasi Anda. Jika Anda menggunakan `Amazon.Lambda.Tools` CLI untuk membangun aplikasi Anda dan Anda tidak mengatur nama assembly menggunakan `AssemblyName` properti dalam `.csproj` file, maka `ASSEMBLY` hanya nama file Anda`.csproj`.
- `TYPE`adalah nama lengkap dari tipe handler, yaitu`NAMESPACE.CLASSNAME`.
- `METHOD`adalah nama metode handler utama dalam kode Anda, yaitu`METHODNAME`.

Untuk contoh kode utama pada halaman ini, jika perakitan diberi nama`ExampleCS`, maka string handler penuh adalah`ExampleCS::ExampleLambda.OrderHandler::HandleRequest`.

Penangan perakitan yang dapat dieksekusi

Anda juga dapat mendefinisikan fungsi Lambda di C # sebagai perakitan yang dapat dieksekusi. Penangan perakitan yang dapat dieksekusi menggunakan fitur pernyataan tingkat atas C#, di mana kompiler menghasilkan `Main()` metode dan menempatkan kode fungsi Anda di dalamnya. Saat menggunakan rakitan yang dapat dieksekusi, runtime Lambda harus di-bootstrap.

Untuk melakukan ini, gunakan `LambdaBootstrapBuilder.Create` metode dalam kode Anda. Input untuk metode ini adalah fungsi handler utama serta serializer Lambda untuk digunakan. Berikut ini menunjukkan contoh handler assembly executable di C #:

```
namespace GetProductHandler;

IDatabaseRepository repo = new DatabaseRepository();

await LambdaBootstrapBuilder.Create<APIGatewayProxyRequest>(Handler, new
 DefaultLambdaJsonSerializer())
    .Build()
    .RunAsync();

async Task<APIGatewayProxyResponse> Handler(APIGatewayProxyRequest apigProxyEvent,
 ILambdaContext context)
{
    var id = apigProxyEvent.PathParameters["id"];
    var databaseRecord = await this.repo.GetById(id);

    return new APIGatewayProxyResponse
    {
        StatusCode = (int)HttpStatusCode.OK,
        Body = JsonSerializer.Serialize(databaseRecord)
```

```
    };
};
```

Di bidang Handler untuk penangan perakitan yang dapat dieksekusi, string handler yang memberi tahu Lambda cara menjalankan kode Anda adalah nama perakitan. Dalam contoh ini, ituGetProductHandler.

Tanda tangan handler yang valid untuk fungsi C #

Di C #, tanda tangan penangan Lambda yang valid mengambil antara 0 dan 2 argumen. Biasanya, tanda tangan handler Anda memiliki dua argumen, seperti yang ditunjukkan pada contoh utama:

```
public async Task<string> HandleRequest(Order order, ILambdaContext context)
```

Saat memberikan dua argumen, argumen pertama harus berupa input peristiwa, dan argumen kedua harus berupa objek konteks Lambda. Kedua argumen bersifat opsional. Misalnya, berikut ini juga merupakan tanda tangan penangan Lambda yang valid di C #:

- `public async Task<string> HandleRequest()`
- `public async Task<string> HandleRequest(Order order)`
- `public async Task<string> HandleRequest(ILambdaContext context)`

Terlepas dari sintaks dasar tanda tangan handler, ada beberapa batasan tambahan:

- Anda tidak dapat menggunakan `unsafe` kata kunci dalam tanda tangan handler. Namun, Anda dapat menggunakan `unsafe` konteks di dalam metode handler dan dependensinya. Untuk informasi selengkapnya, lihat unsafe (referensi C#) di situs web dokumentasi Microsoft.
- Handler tidak dapat menggunakan `params` kata kunci, atau digunakan `ArgIterator` sebagai input atau parameter pengembalian. Kata kunci ini mendukung sejumlah parameter variabel. Jumlah maksimum argumen yang dapat diterima handler Anda adalah dua.
- Handler tidak bisa menjadi metode generik. Dengan kata lain, itu tidak dapat menggunakan parameter tipe generik seperti<T>.
- Lambda tidak mendukung penangan asinkron dengan tanda tangan. `async void`

Konvensi penamaan handler

Penangan Lambda di C# tidak memiliki batasan penamaan yang ketat. Namun, Anda harus memastikan bahwa Anda memberikan string handler yang benar ke Lambda saat Anda menerapkan fungsi Anda. String handler kanan bergantung pada apakah Anda menggunakan handler perpustakaan kelas atau handler perakitan yang dapat dieksekusi.

Meskipun Anda dapat menggunakan nama apa pun untuk handler Anda, nama fungsi di C # umumnya masuk. PascalCase Juga, meskipun nama file tidak perlu cocok dengan nama kelas atau nama handler, umumnya praktik terbaik untuk menggunakan nama file seperti `OrderHandler.cs` jika nama kelas Anda. `OrderHandler` Misalnya, Anda dapat mengubah nama file dalam contoh ini dari `Function.cs` ke`OrderHandler.cs`.

Serialisasi dalam Fungsi Acara C# Lambda

JSON adalah format input yang paling umum dan standar untuk Lambda Event Functions. Dalam contoh ini, fungsi mengharapkan input yang mirip dengan yang berikut:

```
{
    "orderId": "12345",
    "amount": 199.99,
    "item": "Wireless Headphones"
}
```

Di C #, Anda dapat menentukan bentuk peristiwa input yang diharapkan di kelas. Dalam contoh ini, kita mendefinisikan `Order` kelas untuk memodelkan masukan ini:

```
public class Order
{
    public string OrderId { get; set; } = string.Empty;
    public double Amount { get; set; }
    public string Item { get; set; } = string.Empty;
}
```

Jika fungsi Lambda Anda menggunakan jenis input atau output selain `Stream` objek, Anda harus menambahkan pustaka serialisasi ke aplikasi Anda. Anda kemudian dapat mengonversi input JSON menjadi instance dari kelas yang Anda tentukan. Ada dua metode serialisasi untuk fungsi C# di Lambda: serialisasi berbasis refleksi dan serialisasi yang dihasilkan sumber.

Reflection-based serialisasi

AWS menyediakan pustaka pra-bangun yang dapat Anda tambahkan dengan cepat ke aplikasi Anda. Pustaka ini menerapkan serialisasi menggunakan refleksi. Gunakan salah satu paket berikut untuk mengimplementasikan serialisasi berbasis refleksi:

- `Amazon.Lambda.Serialization.SystemTextJson`— Di backend, paket ini digunakan `System.Text.Json` untuk melakukan tugas serialisasi.
- `Amazon.Lambda.Serialization.Json`— Di backend, paket ini digunakan `Newtonsoft.Json` untuk melakukan tugas serialisasi.

Anda juga dapat membuat pustaka serialisasi Anda sendiri dengan mengimplementasikan `ILambdaSerializer` antarmuka, yang tersedia sebagai bagian dari `Amazon.Lambda.Core` perpustakaan. Antarmuka ini mendefinisikan dua metode:

- `T Deserialize<T>(Stream requestStream);`

  Anda menerapkan metode ini untuk deserialisasi payload permintaan dari `Invoke` API ke objek yang diteruskan ke penangan fungsi Lambda Anda.
- `T Serialize<T>(T response, Stream responseStream);`

  Anda menerapkan metode ini untuk membuat serial hasil yang dikembalikan dari penangan fungsi Lambda ke dalam payload respons yang ditampilkan oleh operasi API. `Invoke`

Contoh utama pada halaman ini menggunakan serialisasi berbasis refleksi. Reflection-based serialisasi bekerja di luar kotak dengan AWS Lambda dan tidak memerlukan pengaturan tambahan, menjadikannya pilihan yang baik untuk kesederhanaan. Namun, itu memang membutuhkan lebih banyak penggunaan memori fungsi. Anda mungkin juga melihat latensi fungsi yang lebih tinggi karena refleksi runtime.

Source-generated serialisasi

Dengan serialisasi yang dihasilkan sumber, kode serialisasi dihasilkan pada waktu kompilasi. Ini menghilangkan kebutuhan akan refleksi dan dapat meningkatkan kinerja fungsi Anda. Untuk menggunakan serialisasi yang dihasilkan sumber dalam fungsi Anda, Anda harus melakukan hal berikut:

- Buat kelas parsi baru yang mewarisi dari`JsonSerializerContext`, menambahkan `JsonSerializable` atribut untuk semua jenis yang memerlukan serialisasi atau deserialisasi.
- Konfigurasikan `LambdaSerializer` untuk menggunakan a`SourceGeneratorLambdaJsonSerializer<T>`.
- Perbarui serialisasi manual dan deserialisasi dalam kode aplikasi Anda untuk menggunakan kelas yang baru dibuat.

Contoh berikut menunjukkan bagaimana Anda dapat memodifikasi contoh utama pada halaman ini, yang menggunakan serialisasi berbasis refleksi, untuk menggunakan serialisasi yang dihasilkan sumber sebagai gantinya.

```
using System.Text.Json;
using System.Text.Json.Serialization;

...

public class Order
{
    public string OrderId { get; set; } = string.Empty;
    public double Amount { get; set; }
    public string Item { get; set; } = string.Empty;
}

[JsonSerializable(typeof(Order))]
public partial class OrderJsonContext : JsonSerializerContext {}

public class OrderHandler
{

    ...

    public async Task<string> HandleRequest(string input, ILambdaContext context)
    {

    var order = JsonSerializer.Deserialize(input, OrderJsonContext.Default.Order);

    ...

    }
```

```
}
```

Source-generated serialisasi membutuhkan lebih banyak pengaturan daripada serialisasi berbasis refleksi. Namun, fungsi yang menggunakan sumber yang dihasilkan cenderung menggunakan lebih sedikit memori dan memiliki kinerja yang lebih baik karena pembuatan kode waktu kompilasi. Untuk membantu menghilangkan fungsi cold start, pertimbangkan untuk beralih ke serialisasi yang dihasilkan sumber.

> **Note**
>
> Jika Anda ingin menggunakan kompilasi bawaan sebelumnya (AOT) dengan Lambda, Anda harus menggunakan serialisasi yang dihasilkan sumber.

File-based fungsi

Diperkenalkan di.NET 10, aplikasi berbasis file memungkinkan Anda membangun aplikasi.NET dari satu `.cs` file, tanpa struktur `.csproj` file atau direktori. Lambda mendukung fungsi berbasis file, dimulai dengan .NET 10. Mereka menawarkan cara yang efisien dan ringan untuk membangun Fungsi Acara Lambda di C #.

Cara tercepat untuk mulai membuat fungsi Lambda berbasis file C # adalah dengan menggunakan paket. `Amazon.Lambda.Templates` Untuk memasang paket, jalankan perintah berikut:

```
dotnet new install Amazon.Lambda.Templates
```

Selanjutnya, buat fungsi contoh Lambda berbasis file C #:

```
dotnet new lambda.FileBased -n MyLambdaFunction
```

File-based fungsi menggunakan penangan perakitan yang dapat dieksekusi. Oleh karena itu Anda harus menyertakan `Amazon.Lambda.RuntimeSupport` NuGet paket dan menggunakan `LambdaBootstrapBuilder.Create` metode untuk mendaftarkan fungsi handler .NET untuk jenis acara dan memulai klien runtime .NET Lambda.

File-based fungsi menggunakan .NET Native AOT secara default, yang membutuhkan serialisasi yang dihasilkan sumber. Anda dapat menonaktifkan AOT Asli dengan menentukan `#:property PublishAot=false` di file sumber Anda. Untuk informasi selengkapnya tentang penggunaan AOT Asli di Lambda, lihat. the section called “Kompilasi AOT asli”

Mengakses dan menggunakan objek konteks Lambda

Objek konteks Lambda berisi informasi tentang lingkungan pemanggilan, fungsi, dan eksekusi. Dalam contoh ini, objek konteks bertipe`Amazon.Lambda.Core.ILambdaContext`, dan merupakan argumen kedua dari fungsi pengendali utama.

```
public async Task<string> HandleRequest(Order order, ILambdaContext context) {
    ...
}
```

Objek konteks adalah input opsional. Untuk informasi selengkapnya tentang tanda tangan handler yang diterima valid, lihat. the section called “Tanda tangan handler yang valid untuk fungsi C #”

Objek konteks berguna untuk memproduksi log fungsi ke Amazon CloudWatch. Anda dapat menggunakan `context.getLogger()` metode ini untuk mendapatkan `LambdaLogger` objek untuk logging. Dalam contoh ini, kita dapat menggunakan logger untuk mencatat pesan kesalahan jika pemrosesan gagal karena alasan apa pun:

```
context.Logger.LogError($"Failed to process order: {ex.Message}");
```

Di luar logging, Anda juga dapat menggunakan objek konteks untuk pemantauan fungsi. Untuk informasi selengkapnya tentang objek konteks, lihatthe section called “Konteks”.

Menggunakan SDK untuk .NET v3 di handler Anda

Seringkali, Anda akan menggunakan fungsi Lambda untuk berinteraksi dengan atau membuat pembaruan ke sumber daya lain AWS . Cara paling sederhana untuk berinteraksi dengan sumber daya ini adalah dengan menggunakan SDK untuk .NET v3.

> **Note**
>
> SDK untuk .NET (v2) tidak digunakan lagi. Kami menyarankan Anda hanya menggunakan SDK untuk .NET v3.

Anda dapat menambahkan dependensi SDK ke project menggunakan perintah berikut: `Amazon.Lambda.Tools`

```
dotnet add package <package_name>
```

Misalnya, dalam contoh utama di halaman ini, kita perlu menggunakan Amazon S3 API untuk mengunggah tanda terima ke S3. Kami dapat mengimpor klien SDK Amazon S3 dengan perintah berikut:

```
dotnet add package AWSSDK.S3
```

Perintah ini menambahkan ketergantungan ke proyek Anda. Anda juga harus melihat baris yang mirip dengan yang berikut di `.csproj` file proyek Anda:

```
<PackageReference Include="AWSSDK.S3" Version="3.7.2.18" />
```

Kemudian, impor dependensi langsung di kode C# Anda:

```
using Amazon.S3;
using Amazon.S3.Model;
```

Kode contoh kemudian menginisialisasi klien Amazon S3 (menggunakan rantai penyedia kredensi default) sebagai berikut:

```
private static readonly AmazonS3Client s3Client = new();
```

Dalam contoh ini, kami menginisialisasi klien Amazon S3 kami di luar fungsi pengendali utama untuk menghindari keharusan menginisialisasi setiap kali kami menjalankan fungsi kami. Setelah menginisialisasi klien SDK, Anda dapat menggunakannya untuk berinteraksi dengan layanan lain AWS . Kode contoh memanggil Amazon S3 `PutObject` API sebagai berikut:

```
var putRequest = new PutObjectRequest
{
    BucketName = bucketName,
    Key = key,
    ContentBody = receiptContent,
    ContentType = "text/plain"
};

await s3Client.PutObjectAsync(putRequest);
```

Mengakses variabel lingkungan

Dalam kode handler Anda, Anda dapat mereferensikan variabel lingkungan apa pun dengan menggunakan `System.Environment.GetEnvironmentVariable` metode ini. Dalam contoh ini,

kami mereferensikan variabel `RECEIPT_BUCKET` lingkungan yang ditentukan menggunakan baris kode berikut:

```
string? bucketName = Environment.GetEnvironmentVariable("RECEIPT_BUCKET");
if (string.IsNullOrWhiteSpace(bucketName))
{
    throw new ArgumentException("RECEIPT_BUCKET environment variable is not set");
}
```

Menggunakan status global

Lambda menjalankan kode statis Anda dan konstruktor kelas selama fase inisialisasi sebelum menjalankan fungsi Anda untuk pertama kalinya. Sumber daya yang dibuat selama inisialisasi tetap berada di memori di antara pemanggilan, sehingga Anda dapat menghindari keharusan membuatnya setiap kali Anda menjalankan fungsi Anda.

Dalam kode contoh, kode inisialisasi klien S3 berada di luar metode handler utama. Runtime menginisialisasi klien sebelum fungsi menangani peristiwa pertamanya, yang dapat menyebabkan waktu pemrosesan lebih lama. Peristiwa selanjutnya jauh lebih cepat karena Lambda tidak perlu menginisialisasi klien lagi.

Sederhanakan kode fungsi dengan kerangka kerja Anotasi Lambda

Anotasi Lambda adalah kerangka kerja untuk.NET 8 yang menyederhanakan penulisan fungsi Lambda menggunakan C #. Kerangka kerja Anotasi menggunakan generator sumber untuk menghasilkan kode yang diterjemahkan dari model pemrograman Lambda ke kode yang disederhanakan. Dengan kerangka kerja Anotasi, Anda dapat mengganti banyak kode dalam fungsi Lambda yang ditulis menggunakan model pemrograman biasa. Kode yang ditulis menggunakan kerangka kerja menggunakan ekspresi sederhana yang memungkinkan Anda untuk fokus pada logika bisnis Anda. Lihat Amazon.Lambda.Annotationsdi dokumentasi nuget untuk contoh.

Untuk contoh aplikasi lengkap menggunakan Anotasi Lambda, lihat PhotoAssetManagercontoh di repositori. `awsdocs/aws-doc-sdk-examples` GitHub `Function.cs`File utama dalam `PamApiAnnotations` direktori menggunakan Anotasi Lambda. Sebagai perbandingan, `PamApi` direktori memiliki file setara yang ditulis menggunakan model pemrograman Lambda biasa.

Injeksi ketergantungan dengan kerangka kerja Anotasi Lambda

Anda juga dapat menggunakan kerangka kerja Anotasi Lambda untuk menambahkan injeksi ketergantungan ke fungsi Lambda Anda menggunakan sintaks yang Anda kenal. Saat Anda

menambahkan `[LambdaStartup]` atribut ke `Startup.cs` file, kerangka kerja Anotasi Lambda menghasilkan kode yang diperlukan pada waktu kompilasi.

```
[LambdaStartup]
public class Startup
{
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddSingleton<IDatabaseRepository, DatabaseRepository>();
    }
}
```

Fungsi Lambda Anda dapat menyuntikkan layanan menggunakan injeksi konstruktor atau dengan menyuntikkan ke dalam metode individual menggunakan atribut. `[FromServices]`

```
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer

namespace GetProductHandler;

public class Function
{
    private readonly IDatabaseRepository _repo;

    public Function(IDatabaseRepository repo)
    {
        this._repo = repo;
    }

    [LambdaFunction]
    [HttpApi(LambdaHttpMethod.Get, "/product/{id}")]
    public async Task<Product> FunctionHandler([FromServices] IDatabaseRepository
 repository, string id)
    {
        return await this._repo.GetById(id);
    }
}
```

Praktik terbaik kode untuk Fungsi Acara C# Lambda

Patuhi pedoman dalam daftar berikut untuk menggunakan praktik pengkodean terbaik saat membangun fungsi Lambda Anda:

- Pisahkan handler Lambda dari logika inti Anda. Dengan pendekatan ini, Anda dapat membuat fungsi yang lebih dapat diuji unit.
- Kontrol dependensi dalam paket penerapan fungsi Anda. Lingkungan AWS Lambda eksekusi berisi sejumlah pustaka. Untuk mengaktifkan serangkaian fitur dan pembaruan keamanan terbaru, Lambda memperbarui pustaka ini secara berkala. Pembaruan ini mungkin memperkenalkan perubahan halus pada perilaku fungsi Lambda Anda. Untuk memiliki kendali penuh atas dependensi yang digunakan fungsi Anda, kemas semua dependensi Anda dengan paket deployment Anda.
- Minimalkan kompleksitas dependensi Anda. Utamakan memilih kerangka kerja lebih sederhana yang cepat dimuat dalam memulai lingkungan eksekusi.
- Minimalkan ukuran paket penerapan Anda sesuai kebutuhan runtime-nya. Ini mengurangi jumlah waktu yang diperlukan untuk paket penerapan Anda untuk diunduh dan dibongkar sebelum pemanggilan. Untuk fungsi yang ditulis di .NET, hindari mengunggah seluruh pustaka AWS SDK sebagai bagian dari paket penerapan Anda. Sebagai gantinya, secara selektif bergantung pada modul yang mengambil komponen SDK yang Anda butuhkan (misalnya, DynamoDB, modul SDK Amazon S3, dan pustaka inti Lambda).

Manfaatkan penggunaan kembali lingkungan eksekusi untuk meningkatkan kinerja fungsi Anda. Inisialisasi klien SDK dan koneksi basis data di luar fungsi handler, dan lakukan caching aset statis secara lokal di direktori `/tmp`. Invokasi selanjutnya yang diproses oleh instans yang sama dari fungsi Anda dapat menggunakan kembali sumber daya ini. Ini menghemat biaya dengan mengurangi waktu pengoperasian fungsi.

Untuk menghindari potensi kebocoran data di seluruh invokasi, jangan menggunakan lingkungan eksekusi untuk menyimpan data pengguna, peristiwa, atau informasi lainnya implikasi keamanan. Jika fungsi Anda bergantung pada status yang dapat disenyapkan yang tidak dapat disimpan dalam memori di dalam handler, pertimbangkan untuk membuat fungsi terpisah atau versi terpisah dari fungsi untuk setiap pengguna.

Gunakan arahan keep-alive untuk mempertahankan koneksi yang persisten. Lambda membersihkan koneksi idle dari waktu ke waktu. Mencoba menggunakan ulang koneksi idle saat mengidentifikasi suatu fungsi akan menyebabkan kesalahan koneksi. Untuk mempertahankan koneksi yang

persisten, gunakan arahan tetap aktif yang berkaitan dengan runtime Anda. Sebagai contoh, lihat Menggunakan Kembali Koneksi dengan Keep-Alive in Node.js.

Gunakan variabel lingkungan untuk meneruskan parameter operasional ke fungsi Anda. Misalnya, jika Anda ingin menulis ke bucket Amazon S3 alih-alih melakukan hard-coding nama bucket yang Anda tulis, konfigurasikan nama bucket sebagai variabel lingkungan.

Hindari menggunakan pemanggilan rekursif dalam fungsi Lambda Anda, di mana fungsi memanggil dirinya sendiri atau memulai proses yang dapat memanggil fungsi lagi. Hal ini dapat menyebabkan volume invokasi fungsi yang tidak diinginkan dan peningkatan biaya. Jika Anda melihat volume pemanggilan yang tidak diinginkan, setel konkurensi fungsi cadangan untuk `0` segera membatasi semua pemanggilan ke fungsi, saat Anda memperbarui kode.

Jangan gunakan API non-publik yang tidak terdokumentasi dalam kode fungsi Lambda Anda. Untuk runtime AWS Lambda terkelola, Lambda secara berkala menerapkan pembaruan keamanan dan fungsional ke API internal Lambda. Pembaruan API internal ini mungkin tidak kompatibel ke belakang, yang menyebabkan konsekuensi yang tidak diinginkan seperti kegagalan pemanggilan jika fungsi Anda memiliki ketergantungan pada API non-publik ini. Lihat referensi API untuk daftar API yang tersedia untuk umum.

Tulis kode idempoten. Menulis kode idempoten untuk fungsi Anda memastikan bahwa peristiwa duplikat ditangani dengan cara yang sama. Kode Anda harus memvalidasi peristiwa dengan benar dan menangani peristiwa duplikat dengan anggun. Untuk informasi selengkapnya, lihat Bagaimana cara membuat fungsi Lambda saya idempoten? .

Bangun dan terapkan Fungsi Acara C# Lambda dengan arsip file.zip

Paket penerapan .NET (arsip file.zip) berisi rakitan yang dikompilasi fungsi Anda bersama dengan semua dependensi perakitannya. Paket ini juga berisi file *`proj`*`.deps.json`. Ini memberi sinyal ke runtime .NET semua dependensi fungsi Anda dan *`proj`*`.runtimeconfig.json` file, yang digunakan untuk mengonfigurasi runtime.

Untuk menyebarkan Fungsi Peristiwa Lambda individual, Anda dapat menggunakan CLI Global `Amazon.Lambda.Tools` .NET Lambda. Menggunakan `dotnet lambda deploy-function` perintah secara otomatis membuat paket penyebaran.zip dan menyebarkannya ke Lambda. Namun, kami menyarankan Anda menggunakan kerangka kerja seperti AWS Serverless Application Model (AWS SAM) atau AWS Cloud Development Kit (AWS CDK) untuk menyebarkan aplikasi.NET Anda. AWS

Aplikasi tanpa server biasanya terdiri dari kombinasi fungsi Lambda dan Layanan AWS kerja sama lain yang dikelola untuk melakukan tugas bisnis tertentu. AWS SAM dan AWS CDK menyederhanakan membangun dan menerapkan fungsi Lambda dengan yang lain dalam skala besar. Layanan AWS Spesifikasi AWS SAM template menyediakan sintaks sederhana dan bersih untuk menggambarkan fungsi Lambda, API, izin, konfigurasi, dan sumber daya AWS lain yang membentuk aplikasi tanpa server Anda. Dengan AWS CDKAnda mendefinisikan infrastruktur cloud sebagai kode untuk membantu Anda membangun aplikasi yang andal, terukur, dan hemat biaya di cloud menggunakan bahasa pemrograman modern dan kerangka kerja seperti .NET. Baik AWS CDK dan AWS SAM menggunakan CLI Global .NET Lambda untuk mengemas fungsi Anda.

Meskipun dimungkinkan untuk menggunakan lapisan Lambda dengan fungsi di C # dengan menggunakan CLI CLI.NET Core, kami sarankan untuk tidak melakukannya. Fungsi dalam C# yang menggunakan lapisan secara manual memuat rakitan bersama ke dalam memori selamaFase inisialisasi, yang dapat meningkatkan waktu mulai dingin. Sebagai gantinya, sertakan semua kode bersama pada waktu kompilasi untuk menghindari dampak kinerja pemuatan rakitan saat runtime.

Anda dapat menemukan petunjuk untuk membangun dan menerapkan fungsi.NET Lambda menggunakan AWS SAM, CLI Global Lambda, dan .NET Lambda Global di AWS CDK bagian berikut.

Topik

- Menggunakan CLI Global .NET Lambda
- Menyebarkan Fungsi Acara C# Lambda menggunakan AWS SAM
- Menyebarkan Fungsi Acara C# Lambda menggunakan AWS CDK

- Menyebarkan aplikasi ASP.NET

Menggunakan CLI Global .NET Lambda

.NET CLI dan ekstensi .NET Lambda Global Tools (`Amazon.Lambda.Tools`) menawarkan cara lintas platform untuk membuat. NET-based Aplikasi Lambda, paketkan, dan sebarkan ke Lambda. Di bagian ini, Anda mempelajari cara membuat proyek Lambda.NET baru menggunakan templat .NET CLI dan Amazon Lambda, dan mengemas dan menerapkannya menggunakan `Amazon.Lambda.Tools`

Topik

- Prasyarat
- Membuat proyek .NET menggunakan .NET CLI
- Menerapkan proyek .NET menggunakan .NET CLI
- Menggunakan layer Lambda dengan .NET CLI

Prasyarat

.NET 8 SDK

Jika Anda belum melakukannya, instal SDK .NET 8 dan Runtime.

AWS Amazon.Lambda.Templates Templat proyek .NET

Untuk menghasilkan kode fungsi Lambda Anda, gunakan paket. Amazon.Lambda.Templates NuGet Untuk menginstal paket template ini, jalankan perintah berikut:

```
dotnet new install Amazon.Lambda.Templates
```

AWS Amazon.Lambda.Tools .NET alat CLI Global

Untuk membuat fungsi Lambda Anda, Anda menggunakan Amazon.Lambda.Toolsekstensi.NET Global Tools. Untuk menginstalAmazon.Lambda.Tools, jalankan perintah berikut:

```
dotnet tool install -g Amazon.Lambda.Tools
```

Untuk informasi selengkapnya tentang Amazon.Lambda.Tools ekstensi.NET CLI, lihat AWS Ekstensi untuk repositori.NET CLI di. GitHub

Membuat proyek .NET menggunakan .NET CLI

Di .NET CLI, Anda menggunakan `dotnet new` perintah untuk membuat proyek .NET dari baris perintah. Lambda menawarkan template tambahan menggunakan paket. Amazon.Lambda.Templates NuGet

Setelah menginstal paket ini, jalankan perintah berikut untuk melihat daftar template yang tersedia.

```
dotnet new list
```

Untuk memeriksa rincian tentang templat , gunakan opsi `help`. Misalnya, untuk melihat detail tentang `lambda.EmptyFunction` template, jalankan perintah berikut.

```
dotnet new lambda.EmptyFunction --help
```

Untuk membuat template dasar untuk fungsi.NET Lambda, gunakan template. `lambda.EmptyFunction` Ini menciptakan fungsi sederhana yang mengambil string sebagai input dan mengubahnya menjadi huruf besar menggunakan `ToUpper` metode. Template ini mendukung opsi berikut:

- `--name` – Nama fungsi.
- `--region`— AWS Wilayah untuk membuat fungsi di.
- `--profile`— Nama profil di file AWS SDK untuk .NET kredensil Anda. Untuk mempelajari selengkapnya tentang profil kredensial.NET, lihat Mengonfigurasi AWS kredensil di SDK for AWS .NET Developer Guide.

Dalam contoh ini, kita membuat fungsi kosong baru bernama `myDotnetFunction` menggunakan profil default dan Wilayah AWS pengaturan:

```
dotnet new lambda.EmptyFunction --name myDotnetFunction
```

Perintah ini membuat file dan direktori berikut di direktori proyek Anda.

```
### myDotnetFunction
    ### src
    #   ### myDotnetFunction
    #       ### Function.cs
    #       ### Readme.md
```

```
    #       ### aws-lambda-tools-defaults.json
    #       ### myDotnetFunction.csproj
    ### test
        ### myDotnetFunction.Tests
            ### FunctionTest.cs
            ### myDotnetFunction.Tests.csproj
```

Di bawah direktori `src/myDotnetFunction`, periksa file-file berikut:

- aws-lambda-tools-defaults.json: Ini adalah tempat Anda menentukan opsi baris perintah saat men-deploy fungsi Lambda Anda. Contoh:

```
  "profile" : "default",
  "region" : "us-east-2",
  "configuration" : "Release",
  "function-architecture": "x86_64",
  "function-runtime":"dotnet8",
  "function-memory-size" : 256,
  "function-timeout" : 30,
  "function-handler" : "myDotnetFunction::myDotnetFunction.Function::FunctionHandler"
```

- Function.cs: Kode fungsi penangan Lambda Anda. Ini adalah templat C# yang menyertakan pustaka `Amazon.Lambda.Core` default dan atribut `LambdaSerializer` default. Untuk informasi selengkapnya tentang persyaratan dan opsi serialisasi, lihat Serialisasi dalam Fungsi Acara C# Lambda. Ini juga mencakup fungsi sampel yang dapat Anda edit untuk menerapkan kode fungsi Lambda Anda.

```
using Amazon.Lambda.Core;

// Assembly attribute to enable the Lambda function's JSON input to be converted into
 a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializ

namespace myDotnetFunction;

public class Function
{

    /// <summary>
    /// A simple function that takes a string and does a ToUpper
    /// </summary#
```

```
    /// <param name="input"></param>
    /// <param name="context"></param>
    /// <returns></returns>
    public string FunctionHandler(string input, ILambdaContext context)
    {
        return input.ToUpper();
    }
}
```

- my DotnetFunction.csproj: File MSBuild yang mencantumkan file dan rakitan yang terdiri dari aplikasi Anda.

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <TargetFramework>net8.0</TargetFramework>
    <ImplicitUsings>enable</ImplicitUsings>
    <Nullable>enable</Nullable>
    <GenerateRuntimeConfigurationFiles>true</GenerateRuntimeConfigurationFiles>
    <AWSProjectType>Lambda</AWSProjectType>
    <!-- This property makes the build directory similar to a publish directory and
 helps the AWS .NET Lambda Mock Test Tool find project dependencies. -->
    <CopyLocalLockFileAssemblies>true</CopyLocalLockFileAssemblies>
    <!-- Generate ready to run images during publishing to improve cold start time.
 -->
    <PublishReadyToRun>true</PublishReadyToRun>
  </PropertyGroup>
  <ItemGroup>
    <PackageReference Include="Amazon.Lambda.Core" Version="2.2.0" />
    <PackageReference Include="Amazon.Lambda.Serialization.SystemTextJson"
 Version="2.4.0" />
  </ItemGroup>
</Project>
```

- Readme: Gunakan file ini untuk mendokumentasikan fungsi Lambda Anda.

Di bawah direktori `myfunction/test`, periksa file-file berikut:

- my DotnetFunction.Tests.csproj: Seperti disebutkan sebelumnya, ini adalah file MSBuild yang mencantumkan file dan rakitan yang terdiri dari proyek pengujian Anda. Perhatikan juga bahwa ini menyertakan `Amazon.Lambda.Core` pustaka, sehingga Anda dapat dengan mulus mengintegrasikan template Lambda apa pun yang diperlukan untuk menguji fungsi Anda.

```
<Project Sdk="Microsoft.NET.Sdk">
   ...

    <PackageReference Include="Amazon.Lambda.Core" Version="2.2.0 " />
   ...
```

- FunctionTest.cs: File template kode C # yang sama yang disertakan dalam `src` direktori. Edit file ini untuk mencerminkan kode produksi fungsi Anda dan mengujinya sebelum mengunggah fungsi Lambda Anda ke lingkungan produksi.

```
using Xunit;
using Amazon.Lambda.Core;
using Amazon.Lambda.TestUtilities;

using MyFunction;

namespace MyFunction.Tests
{
    public class FunctionTest
    {
        [Fact]
        public void TestToUpperFunction()
        {

            // Invoke the lambda function and confirm the string was upper cased.
            var function = new Function();
            var context = new TestLambdaContext();
            var upperCase = function.FunctionHandler("hello world", context);

            Assert.Equal("HELLO WORLD", upperCase);
        }
    }
}
```

Menerapkan proyek .NET menggunakan .NET CLI

Untuk membangun paket penyebaran Anda dan menyebarkannya ke Lambda, Anda menggunakan alat CLI. `Amazon.Lambda.Tools` Untuk menerapkan fungsi Anda dari file yang Anda buat di langkah sebelumnya, pertama-tama navigasikan ke folder yang berisi `.csproj` file fungsi Anda.

```
cd myDotnetFunction/src/myDotnetFunction
```

Untuk menyebarkan kode Anda ke Lambda sebagai paket penyebaran .zip, jalankan perintah berikut. Pilih nama fungsi Anda sendiri.

```
dotnet lambda deploy-function myDotnetFunction
```

Selama penyebaran, wizard meminta Anda untuk memilih file. the section called “Peran eksekusi (izin untuk fungsi untuk mengakses sumber daya lain)” Untuk contoh ini, pilih`lambda_basic_role`.

Setelah Anda menerapkan fungsi Anda, Anda dapat mengujinya di cloud menggunakan `dotnet lambda invoke-function` perintah. Untuk contoh kode dalam `lambda.EmptyFunction` template, Anda dapat menguji fungsi Anda dengan meneruskan string menggunakan `--payload` opsi.

```
dotnet lambda invoke-function myDotnetFunction --payload "Just checking if everything
 is OK"
```

Jika fungsi Anda telah berhasil di-deploy, Anda akan melihat output yang mirip dengan berikut ini.

```
dotnet lambda invoke-function myDotnetFunction --payload "Just checking if everything
 is OK"
Amazon Lambda Tools for .NET Core applications (5.8.0)
Project Home: https://github.com/aws/aws-extensions-for-dotnet-cli, https://github.com/
aws/aws-lambda-dotnet

Payload:
"JUST CHECKING IF EVERYTHING IS OK"

Log Tail:
START RequestId: id Version: $LATEST
END RequestId: id
REPORT RequestId: id  Duration: 0.99 ms       Billed Duration: 1 ms         Memory
 Size: 256 MB     Max Memory Used: 12 MB
```

Menggunakan layer Lambda dengan .NET CLI

> **Note**
>
> Meskipun dimungkinkan untuk menggunakan lapisan dengan fungsi di .NET, kami sarankan untuk tidak melakukannya. Fungsi di .NET yang menggunakan lapisan secara manual memuat rakitan bersama ke dalam memori selama `Init` fase, yang dapat meningkatkan waktu mulai dingin. Sebagai gantinya, sertakan semua kode bersama pada waktu kompilasi untuk memanfaatkan pengoptimalan bawaan compiler.NET.

.NET CLI mendukung perintah untuk membantu Anda mempublikasikan lapisan dan menerapkan fungsi C# yang menggunakan lapisan. Untuk memublikasikan layer ke bucket Amazon S3 tertentu, jalankan perintah berikut di direktori yang sama dengan file Anda `.csproj`:

```
dotnet lambda publish-layer <layer_name> --layer-type runtime-package-store --s3-
bucket <s3_bucket_name>
```

Kemudian, ketika Anda menerapkan fungsi Anda menggunakan .NET CLI, tentukan layer ARN konsumsi dalam perintah berikut:

```
dotnet lambda deploy-function <function_name> --function-layers arn:aws:lambda:us-
east-1:123456789012:layer:layer-name:1
```

Untuk contoh lengkap fungsi Hello World, lihat sampel blank-csharp-with-layer.

Menyebarkan Fungsi Acara C# Lambda menggunakan AWS SAM

The AWS Serverless Application Model (AWS SAM) adalah toolkit yang membantu merampingkan proses membangun dan menjalankan aplikasi tanpa server. AWS Anda menentukan sumber daya untuk aplikasi Anda dalam template YAMM atau JSON dan menggunakan antarmuka baris AWS SAM perintah (AWS SAM CLI) untuk membangun, mengemas, dan menyebarkan aplikasi Anda. Saat Anda membuat fungsi Lambda dari AWS SAM template, AWS SAM secara otomatis membuat paket penerapan .zip atau gambar kontainer dengan kode fungsi Anda dan dependensi apa pun yang Anda tentukan. AWS SAM kemudian menyebarkan fungsi Anda menggunakan CloudFormation tumpukan. Untuk mempelajari lebih lanjut cara menggunakan AWS SAM untuk membangun dan menerapkan fungsi Lambda, lihat Memulai di Panduan AWS SAMAWS Serverless Application Model Pengembang.

Langkah-langkah berikut menunjukkan kepada Anda cara mengunduh, membangun, dan menyebarkan contoh aplikasi.NET Hello World menggunakan AWS SAM. Contoh aplikasi ini menggunakan fungsi Lambda dan endpoint Amazon API Gateway untuk mengimplementasikan backend API dasar. Saat Anda mengirim permintaan HTTP GET ke titik akhir API Gateway, API Gateway akan memanggil fungsi Lambda Anda. Fungsi mengembalikan pesan “hello world”, bersama dengan alamat IP dari instance fungsi Lambda yang memproses permintaan Anda.

Saat Anda membangun dan menerapkan aplikasi Anda menggunakan AWS SAM, di belakang layar AWS SAM CLI menggunakan perintah untuk mengemas `dotnet lambda package` bundel kode fungsi Lambda individual.

Prasyarat

.NET 8 SDK

Instal .NET 8 SDK dan Runtime.

AWS SAM CLI versi 1.39 atau yang lebih baru

Untuk mempelajari cara menginstal AWS SAM CLI versi terbaru, lihat Menginstal CLI AWS SAM.

Menyebarkan sampel AWS SAM application

1. Inisialisasi aplikasi menggunakan template Hello world .NET menggunakan perintah berikut.

```
sam init --app-template hello-world --name sam-app \
--package-type Zip --runtime dotnet8
```

Perintah ini membuat file dan direktori berikut di direktori proyek Anda.

```
### sam-app
    ### README.md
    ### events
    #   ### event.json
    ### omnisharp.json
    ### samconfig.toml
    ### src
    #   ### HelloWorld
    #       ### Function.cs
    #       ### HelloWorld.csproj
    #       ### aws-lambda-tools-defaults.json
```

```
    ### template.yaml
    ### test
        ### HelloWorld.Test
            ### FunctionTest.cs
            ### HelloWorld.Tests.csproj
```

2. Arahkan ke direktori yang berisi file`template.yaml file`. File ini adalah tempate yang mendefinisikan AWS resource untuk aplikasi Anda, termasuk fungsi Lambda Anda dan API Gateway API.

```
cd sam-app
```

3. Untuk membangun sumber aplikasi Anda, jalankan perintah berikut.

```
sam build
```

4. Untuk menyebarkan aplikasi Anda AWS, jalankan perintah berikut.

```
sam deploy --guided
```

Perintah ini mengemas dan menyebarkan aplikasi Anda dengan serangkaian prompt berikut. Untuk menerima opsi default, tekan Enter.

> **Note**
>
> Karena HelloWorldFunction mungkin tidak memiliki otorisasi yang ditentukan, apakah ini baik-baik saja? , pastikan untuk masuky.

- Nama Stack: Nama tumpukan yang akan digunakan. CloudFormation Nama ini harus unik untuk Anda Akun AWS dan Wilayah AWS.
- Wilayah AWS: Wilayah AWS Anda ingin men-deploy aplikasi Anda ke.
- Konfirmasikan perubahan sebelum menerapkan: Pilih ya untuk meninjau set perubahan apa pun secara manual sebelum AWS SAM menerapkan perubahan aplikasi. Jika Anda memilih tidak, AWS SAM CLI secara otomatis menyebarkan perubahan aplikasi.
- Izinkan pembuatan peran SAM CLI IAM: Banyak AWS SAM templat, termasuk Hello world dalam contoh ini, buat peran AWS Identity and Access Management (IAM) untuk memberikan izin fungsi Lambda Anda untuk mengakses yang lain. Layanan AWS Pilih Ya

untuk memberikan izin untuk menyebarkan CloudFormation tumpukan yang membuat atau memodifikasi peran IAM.

- Nonaktifkan rollback: Secara default, jika AWS SAM menemukan kesalahan selama pembuatan atau penyebaran tumpukan Anda, itu akan memutar tumpukan kembali ke versi sebelumnya. Pilih Tidak untuk menerima default ini.
- HelloWorldFunction mungkin tidak memiliki otorisasi yang ditentukan, apakah ini baik-baik saja: Entery.
- Simpan argumen ke samconfig.toml: Pilih ya untuk menyimpan pilihan konfigurasi Anda. Di masa mendatang, Anda dapat menjalankan ulang `sam deploy` tanpa parameter untuk menerapkan perubahan pada aplikasi Anda.

5. Ketika penerapan aplikasi Anda selesai, CLI mengembalikan Amazon Resource Name (ARN) dari fungsi Hello World Lambda dan peran IAM yang dibuat untuknya. Ini juga menampilkan titik akhir API Gateway API Anda. Untuk menguji aplikasi Anda, buka titik akhir di browser. Anda akan melihat respons yang mirip dengan yang berikut ini.

```
{"message":"hello world","location":"34.244.135.203"}
```

6. Untuk menghapus sumber daya Anda, jalankan perintah berikut. Perhatikan bahwa titik akhir API yang Anda buat adalah titik akhir publik yang dapat diakses melalui internet. Kami menyarankan Anda menghapus titik akhir ini setelah pengujian.

```
sam delete
```

Langkah selanjutnya

Untuk mempelajari selengkapnya tentang penggunaan AWS SAM untuk membangun dan menerapkan fungsi Lambda menggunakan.NET, lihat sumber daya berikut:

- Panduan Pengembang AWS Serverless Application Model (AWS SAM)
- Membangun Aplikasi.NET Tanpa Server dengan AWS Lambda dan SAM CLI

Menyebarkan Fungsi Acara C# Lambda menggunakan AWS CDK

AWS Cloud Development Kit (AWS CDK) Ini adalah kerangka pengembangan perangkat lunak open-source untuk mendefinisikan infrastruktur cloud sebagai kode dengan bahasa pemrograman modern

dan kerangka kerja seperti .NET. AWS CDK proyek dijalankan untuk menghasilkan CloudFormation template yang kemudian digunakan untuk menyebarkan kode Anda.

Untuk membangun dan menyebarkan contoh aplikasi Hello world .NET menggunakan AWS CDK, ikuti petunjuk di bagian berikut. Aplikasi sampel mengimplementasikan backend API dasar yang terdiri dari titik akhir API Gateway dan fungsi Lambda. API Gateway memanggil fungsi Lambda saat Anda mengirim permintaan HTTP GET ke titik akhir. Fungsi mengembalikan pesan Hello world, bersama dengan alamat IP dari instance Lambda yang memproses permintaan Anda.

Prasyarat

.NET 8 SDK

Instal .NET 8 SDK dan Runtime.

AWS CDK versi 2

Untuk mempelajari cara menginstal versi terbaru, AWS CDK lihat Memulai dengan AWS CDK di Panduan Pengembang AWS Cloud Development Kit (AWS CDK) v2.

Menyebarkan sampel AWS CDK application

1. Buat direktori proyek untuk aplikasi sampel dan navigasikan ke dalamnya.

```
mkdir hello-world
cd hello-world
```

2. Menginisialisasi AWS CDK aplikasi baru dengan menjalankan perintah berikut.

```
cdk init app --language csharp
```

Perintah membuat file dan direktori berikut di direktori proyek Anda

```
### README.md
### cdk.json
### src
    ### HelloWorld
    #   ### GlobalSuppressions.cs
    #   ### HelloWorld.csproj
    #   ### HelloWorldStack.cs
    #   ### Program.cs
```

```
    ### HelloWorld.sln
```

3. Buka `src` direktori dan buat fungsi Lambda baru menggunakan .NET CLI. Ini adalah fungsi yang Anda terapkan menggunakan. AWS CDK Dalam contoh ini, Anda membuat fungsi Hello world bernama `HelloWorldLambda` menggunakan `lambda.EmptyFunction` template.

```
cd src
dotnet new lambda.EmptyFunction -n HelloWorldLambda
```

Setelah langkah ini, struktur direktori Anda di dalam direktori proyek Anda akan terlihat seperti berikut.

```
### README.md
### cdk.json
### src
    ### HelloWorld
    #   ### GlobalSuppressions.cs
    #   ### HelloWorld.csproj
    #   ### HelloWorldStack.cs
    #   ### Program.cs
    ### HelloWorld.sln
    ### HelloWorldLambda
        ### src
        #   ### HelloWorldLambda
        #       ### Function.cs
        #       ### HelloWorldLambda.csproj
        #       ### Readme.md
        #       ### aws-lambda-tools-defaults.json
        ### test
            ### HelloWorldLambda.Tests
                ### FunctionTest.cs
                ### HelloWorldLambda.Tests.csproj
```

4. Buka `HelloWorldStack.cs` file dari `src/HelloWorld` direktori. Ganti isi file dengan kode berikut.

```
using Amazon.CDK;
using Amazon.CDK.AWS.Lambda;
using Amazon.CDK.AWS.Logs;
using Constructs;

namespace CdkTest
```

```
{
    public class HelloWorldStack : Stack
    {
        internal HelloWorldStack(Construct scope, string id, IStackProps props =
 null) : base(scope, id, props)
        {
            var buildOption = new BundlingOptions()
            {
                Image = Runtime.DOTNET_8.BundlingImage,
                User = "root",
                OutputType = BundlingOutput.ARCHIVED,
                Command = new string[]{
            "/bin/sh",
                "-c",
                " dotnet tool install -g Amazon.Lambda.Tools"+
                " && dotnet build"+
                " && dotnet lambda package --output-package /asset-output/
function.zip"
                }
            };

             var helloWorldLambdaFunction = new Function(this,
 "HelloWorldFunction", new FunctionProps
            {
                Runtime = Runtime.DOTNET_8,
                MemorySize = 1024,
                LogRetention = RetentionDays.ONE_DAY,
                Handler =
 "HelloWorldLambda::HelloWorldLambda.Function::FunctionHandler",
                Code = Code.FromAsset("./src/HelloWorldLambda/src/
HelloWorldLambda", new Amazon.CDK.AWS.S3.Assets.AssetOptions
                {
                    Bundling = buildOption
                }),
            });
        }
    }
}
```

Ini adalah kode untuk mengkompilasi dan menggabungkan kode aplikasi, serta definisi fungsi Lambda itu sendiri. `BundlingOptions` objek memungkinkan file zip dibuat, bersama dengan serangkaian perintah yang digunakan untuk menghasilkan konten file zip. Dalam hal ini, `dotnet lambda package` perintah digunakan untuk mengkompilasi dan menghasilkan file zip.

5. Untuk menyebarkan aplikasi Anda, jalankan perintah berikut.

```
cdk deploy
```

6. Memanggil fungsi Lambda yang Anda gunakan menggunakan .NET Lambda CLI.

```
dotnet lambda invoke-function HelloWorldFunction -p "hello world"
```

7. Setelah selesai menguji, Anda dapat menghapus sumber daya yang Anda buat, kecuali jika Anda ingin mempertahankannya. Jalankan perintah berikut untuk menghapus sumber daya Anda.

```
cdk destroy
```

Langkah selanjutnya

Untuk mempelajari selengkapnya tentang penggunaan AWS CDK untuk membangun dan menerapkan fungsi Lambda menggunakan.NET, lihat sumber daya berikut:

- Bekerja dengan AWS CDK di C #
- Membangun, mengemas, dan mempublikasikan fungsi.NET C # Lambda dengan CDK AWS

Menyebarkan aplikasi ASP.NET

Selain menghosting fungsi berbasis acara, Anda juga dapat menggunakan.NET dengan Lambda Event Functions untuk meng-host aplikasi ringan. ASP.NET Anda dapat membangun dan menyebarkan ASP.NET aplikasi menggunakan `Amazon.Lambda.AspNetCoreServer` NuGet paket. Di bagian ini, Anda mempelajari cara menerapkan API ASP.NET web ke Lambda menggunakan tooling.NET Lambda CLI.

Topik

- Prasyarat
- Menerapkan API ASP.NET Web ke Lambda
- Menerapkan API ASP.NET minimal ke Lambda

Prasyarat

.NET 8 SDK

Instal .NET 8 SDK dan ASP.NET Core Runtime.

Amazon.Lambda.Tools

Untuk membuat fungsi Lambda Anda, Anda menggunakan Amazon.Lambda.Toolsekstensi.NET Global Tools. Untuk menginstalAmazon.Lambda.Tools, jalankan perintah berikut:

```
dotnet tool install -g Amazon.Lambda.Tools
```

Untuk informasi selengkapnya tentang Amazon.Lambda.Tools ekstensi.NET CLI, lihat AWS Ekstensi untuk repositori.NET CLI di. GitHub

Amazon.Lambda.Templates

Untuk menghasilkan kode fungsi Lambda Anda, gunakan paket. Amazon.Lambda.Templates NuGet Untuk menginstal paket template ini, jalankan perintah berikut:

```
dotnet new --install Amazon.Lambda.Templates
```

Menerapkan API ASP.NET Web ke Lambda

Untuk menggunakan API web ASP.NET, Anda dapat menggunakan template .NET Lambda untuk membuat proyek API web baru. Gunakan perintah berikut untuk menginisialisasi proyek API ASP.NET web baru. Dalam perintah contoh, kami memberi nama proyek`AspNetOnLambda`.

```
dotnet new serverless.AspNetCoreWebAPI -n AspNetOnLambda
```

Perintah ini membuat file dan direktori berikut di direktori proyek Anda.

```
.
### AspNetOnLambda
    ### src
    #   ### AspNetOnLambda
    #       ### AspNetOnLambda.csproj
    #       ### Controllers
    #       #   ### ValuesController.cs
    #       ### LambdaEntryPoint.cs
    #       ### LocalEntryPoint.cs
```

```
    #       ### Readme.md
    #       ### Startup.cs
    #       ### appsettings.Development.json
    #       ### appsettings.json
    #       ### aws-lambda-tools-defaults.json
    #       ### serverless.template
    ### test
        ### AspNetOnLambda.Tests
            ### AspNetOnLambda.Tests.csproj
            ### SampleRequests
            #   ### ValuesController-Get.json
            ### ValuesControllerTests.cs
            ### appsettings.json
```

Ketika Lambda memanggil fungsi Anda, titik masuk yang digunakannya adalah file. `LambdaEntryPoint.cs` File yang dibuat oleh template.NET Lambda berisi kode berikut.

```
namespace AspNetOnLambda;

public class LambdaEntryPoint : Amazon.Lambda.AspNetCoreServer.APIGatewayProxyFunction
{
    protected override void Init(IWebHostBuilder builder)
    {
        builder
            .UseStartup#Startup#();
    }

    protected override void Init(IHostBuilder builder)
    {
    }
}
```

Titik masuk yang digunakan oleh Lambda harus mewarisi dari salah satu dari tiga kelas dasar dalam paket. `Amazon.Lambda.AspNetCoreServer` Ketiga kelas dasar ini adalah:

- `APIGatewayProxyFunction`
- `APIGatewayHttpApiV2ProxyFunction`
- `ApplicationLoadBalancerFunction`

Kelas default yang digunakan saat Anda membuat `LambdaEntryPoint.cs` file menggunakan template.NET Lambda yang disediakan adalah. `APIGatewayProxyFunction` Kelas dasar yang

Anda gunakan dalam fungsi Anda bergantung pada lapisan API mana yang berada di depan fungsi Lambda Anda.

Masing-masing dari tiga kelas dasar berisi metode publik bernama`FunctionHandlerAsync`. Nama metode ini merupakan bagian dari string handler yang digunakan Lambda untuk memanggil fungsi Anda. `FunctionHandlerAsync`Metode ini mengubah payload peristiwa masuk ke ASP.NET format yang benar dan ASP.NET respons kembali ke muatan respons Lambda. Untuk `AspNetOnLambda` proyek contoh yang ditampilkan, string handler adalah sebagai berikut.

```
AspNetOnLambda::AspNetOnLambda.LambdaEntryPoint::FunctionHandlerAsync
```

Untuk menerapkan API ke Lambda, jalankan perintah berikut untuk menavigasi ke direktori yang berisi file kode sumber Anda dan menerapkan fungsi Anda menggunakan. CloudFormation

```
cd AspNetOnLambda/src/AspNetOnLambda
dotnet lambda deploy-serverless
```

> **Tip**
>
> Saat Anda menerapkan API menggunakan **`dotnet lambda deploy-serverless`** perintah, CloudFormation beri nama pada fungsi Lambda Anda berdasarkan nama tumpukan yang Anda tentukan selama penerapan. Untuk memberikan nama khusus pada fungsi Lambda Anda, edit `serverless.template` file untuk menambahkan `FunctionName` properti ke sumber daya. `AWS::Serverless::Function` Lihat Jenis nama di Panduan CloudFormation Pengguna untuk mempelajari lebih lanjut.

Menerapkan API ASP.NET minimal ke Lambda

Untuk menerapkan API ASP.NET minimal ke Lambda, Anda dapat menggunakan template.NET Lambda untuk membuat proyek API minimal yang baru. Gunakan perintah berikut untuk menginisialisasi proyek API minimal yang baru. Dalam contoh ini, kami memberi nama proyek`MinimalApiOnLambda`.

```
dotnet new serverless.AspNetCoreMinimalAPI -n MinimalApiOnLambda
```

Perintah membuat file dan direktori berikut di direktori proyek Anda.

```
### MinimalApiOnLambda
    ### src
        ### MinimalApiOnLambda
            ### Controllers
            #   ### CalculatorController.cs
            ### MinimalApiOnLambda.csproj
            ### Program.cs
            ### Readme.md
            ### appsettings.Development.json
            ### appsettings.json
            ### aws-lambda-tools-defaults.json
            ### serverless.template
```

`Program.cs`File berisi kode berikut.

```
var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddControllers();

// Add AWS Lambda support. When application is run in Lambda Kestrel is swapped out as
 the web server with Amazon.Lambda.AspNetCoreServer. This
// package will act as the webserver translating request and responses between the
 Lambda event source and ASP.NET Core.
builder.Services.AddAWSLambdaHosting(LambdaEventSource.RestApi);

var app = builder.Build();


app.UseHttpsRedirection();
app.UseAuthorization();
app.MapControllers();

app.MapGet("/", () => "Welcome to running ASP.NET Core Minimal API on AWS Lambda");

app.Run();
```

Untuk mengonfigurasi API minimal agar berjalan di Lambda, Anda mungkin perlu mengedit kode ini agar permintaan dan tanggapan antara Lambda dan ASP.NET Core diterjemahkan dengan benar. Secara default, fungsi dikonfigurasi untuk sumber peristiwa REST API. Untuk HTTP API atau penyeimbang beban aplikasi, ganti `(LambdaEventSource.RestApi)` dengan salah satu opsi berikut:

- `(LambdaEventSource.HttpAPi)`
- `(LambdaEventSource.ApplicationLoadBalancer)`

Untuk menerapkan API minimal Anda ke Lambda, jalankan perintah berikut untuk menavigasi ke direktori yang berisi file kode sumber Anda dan menerapkan fungsi Anda menggunakan. CloudFormation

```
cd MinimalApiOnLambda/src/MinimalApiOnLambda
dotnet lambda deploy-serverless
```

Bekerja dengan lapisan untuk .NET Lambda Event Functions

Kami tidak menyarankan menggunakan lapisan untuk mengelola dependensi untuk Fungsi Peristiwa Lambda yang ditulis dalam.NET. Ini karena .NET adalah bahasa yang dikompilasi, dan fungsi Anda masih harus memuat rakitan bersama secara manual ke dalam memori selama fase Init, yang dapat meningkatkan waktu mulai dingin. Menggunakan lapisan tidak hanya mempersulit proses penerapan, tetapi juga mencegah Anda memanfaatkan pengoptimalan kompiler bawaan.

Untuk menggunakan dependensi eksternal dengan handler.NET Anda, sertakan mereka langsung dalam paket penerapan Anda pada waktu kompilasi. Dengan demikian, Anda menyederhanakan proses penerapan dan juga memanfaatkan pengoptimalan compiler.NET bawaan. Untuk contoh cara mengimpor dan menggunakan dependensi seperti NuGet paket dalam fungsi Anda, lihat. the section called “Handler”

Menyebarkan Fungsi Event .NET Lambda dengan gambar kontainer

Ada tiga cara untuk membangun image kontainer untuk Fungsi Event .NET Lambda:

- Menggunakan sebuah AWS gambar dasar untuk .NET

  Gambar AWS dasar dimuat sebelumnya dengan runtime bahasa, klien antarmuka runtime untuk mengelola interaksi antara Lambda dan kode fungsi Anda, dan emulator antarmuka runtime untuk pengujian lokal.

- Menggunakan sebuah AWS OS-only gambar dasar

  AWS OS-only image dasar berisi distribusi Amazon Linux dan emulator antarmuka runtime. Gambar-gambar ini biasanya digunakan untuk membuat gambar kontainer untuk bahasa yang dikompilasi, seperti Go dan Rust, dan untuk versi bahasa atau bahasa yang Lambda tidak menyediakan gambar dasar, seperti Node.js 19. Anda juga dapat menggunakan gambar OS-only dasar untuk mengimplementasikan runtime kustom. Untuk membuat gambar kompatibel dengan Lambda, Anda harus menyertakan klien antarmuka runtime untuk.NET dalam gambar.

- Menggunakan gambar AWS non-dasar

  Anda dapat menggunakan gambar dasar alternatif dari registri kontainer lain, seperti Alpine Linux atau Debian. Anda juga dapat menggunakan gambar kustom yang dibuat oleh organisasi Anda. Untuk membuat gambar kompatibel dengan Lambda, Anda harus menyertakan klien antarmuka runtime untuk.NET dalam gambar.

> **Tip**
>
> Untuk mengurangi waktu yang dibutuhkan agar fungsi kontainer Lambda menjadi aktif, lihat Menggunakan build multi-tahap dalam dokumentasi Docker. Untuk membuat gambar kontainer yang efisien, ikuti Praktik terbaik untuk menulis Dockerfiles.

Halaman ini menjelaskan cara membuat, menguji, dan menyebarkan gambar kontainer untuk Lambda.

Topik

- AWS gambar dasar untuk .NET
- Menggunakan sebuah AWS gambar dasar untuk .NET

- Menggunakan gambar dasar alternatif dengan klien antarmuka runtime

AWS gambar dasar untuk .NET

AWS menyediakan gambar dasar berikut untuk .NET:

Tanda	Waktu berjalan	Sistem operasi	Dockerfile	penghentian
10	.NET 10	Amazon Linux 2023	Dockerfile untuk .NET 10 di GitHub	Nov 14, 2028
9	.NET 9	Amazon Linux 2023	Dockerfile untuk .NET 9 di GitHub	Nov 10, 2026
8	.NET 8	Amazon Linux 2023	Dockerfile untuk .NET 8 di GitHub	Nov 10, 2026

Repositori Amazon ECR: gallery.ecr. aws/lambda/dotnet

Menggunakan sebuah AWS gambar dasar untuk .NET

Prasyarat

Untuk menyelesaikan langkah-langkah di bagian ini, Anda harus memiliki hal-hal berikut:

- .NET SDK — Langkah-langkah berikut menggunakan gambar dasar.NET 8. Pastikan versi.NET Anda cocok dengan versi gambar dasar yang Anda tentukan di Dockerfile Anda.
- Docker (versi minimum 25.0.0)
- Plugin buildx Docker.

Membuat dan menyebarkan gambar menggunakan gambar dasar

Dalam langkah-langkah berikut, Anda menggunakan Amazon.Lambda.Templatesdan Amazon.Lambda.Toolsmembuat proyek .NET. Kemudian, Anda membuat gambar Docker, mengunggah gambar ke Amazon ECR, dan menerapkannya ke fungsi Lambda.

1. Instal paket Amazon.Lambda.Templates NuGet .

```
dotnet new install Amazon.Lambda.Templates
```

2. Buat proyek.NET menggunakan `lambda.image.EmptyFunction` template.

```
dotnet new lambda.image.EmptyFunction --name MyFunction --region us-east-1
```

File proyek disimpan dalam *MyFunction*`/src/`*MyFunction* direktori:

- aws-lambda-tools-defaults.json: Menentukan opsi baris perintah untuk menerapkan fungsi Lambda Anda.
- Function.cs: Kode fungsi penangan Lambda Anda. Ini adalah template C # yang mencakup `Amazon.Lambda.Core` pustaka default dan `LambdaSerializer` atribut default. Untuk informasi selengkapnya tentang persyaratan dan opsi serialisasi, lihatSerialisasi dalam Fungsi Acara C# Lambda. Anda dapat menggunakan kode yang disediakan untuk pengujian, atau menggantinya dengan kode Anda sendiri.
- MyFunction.csproj: File proyek.NET, yang mencantumkan file dan rakitan yang terdiri dari aplikasi Anda.
- Dockerfile: Anda dapat menggunakan Dockerfile yang disediakan untuk pengujian, atau menggantinya dengan milik Anda sendiri. Jika Anda menggunakan sendiri, pastikan untuk:
  - Mengatur `FROM` properti ke URI dari gambar dasar. Gambar dasar dan `TargetFramework` dalam `MyFunction.csproj` file harus menggunakan versi.NET yang sama. Misalnya, untuk menggunakan .NET 9:
    - Dockerfile: `FROM` *public.ecr.aws/lambda/dotnet:9*
    - MyFunction.csproj: `<TargetFramework>`*net9.0*`</TargetFramework>`
  - Atur `CMD` argumen ke penangan fungsi Lambda. Ini harus cocok dengan `image-command` in`aws-lambda-tools-defaults.json`.

3. Instal Alat Amazon.Lambda.Tools Global.NET.

```
dotnet tool install -g Amazon.Lambda.Tools
```

Jika Amazon.Lambda.Tools sudah terinstal, pastikan Anda memiliki versi terbaru.

```
dotnet tool update -g Amazon.Lambda.Tools
```

4. Ubah direktori ke*MyFunction*`/src/`*MyFunction*, jika Anda belum ada di sana.

```
cd src/MyFunction
```

5. Gunakan Amazon.Lambda.Tools untuk membuat image Docker, dorong ke repositori Amazon ECR baru, dan gunakan fungsi Lambda.

   Untuk `--function-role`, tentukan nama peran—bukan Amazon Resource Name (ARN) —dari peran eksekusi untuk fungsi tersebut. Misalnya, `lambda-role`.

   ```
   dotnet lambda deploy-function MyFunction --function-role lambda-role
   ```

   Untuk informasi selengkapnya tentang Amazon.Lambda.Tools .NET Global Tool, lihat AWS Ekstensi untuk repositori.NET CLI di. GitHub

6. Memanggil fungsi.

   ```
   dotnet lambda invoke-function MyFunction --payload "Testing the function"
   ```

   Jika semuanya berhasil, Anda melihat respons yang mirip dengan yang berikut:

   ```
   Payload:
   {"Lower":"testing the function","Upper":"TESTING THE FUNCTION"}

   Log Tail:
   INIT_REPORT Init Duration: 9999.81 ms   Phase: init     Status: timeout
   START RequestId: 12378346-f302-419b-b1f2-deaa1e8423ed Version: $LATEST
   END RequestId: 12378346-f302-419b-b1f2-deaa1e8423ed
   REPORT RequestId: 12378346-f302-419b-b1f2-deaa1e8423ed  Duration: 3173.06 ms
    Billed Duration: 3174 ms        Memory Size: 512 MB     Max Memory Used: 24 MB
   ```

7. Hapus fungsi Lambda.

   ```
   dotnet lambda delete-function MyFunction
   ```

Menggunakan gambar dasar alternatif dengan klien antarmuka runtime

Jika Anda menggunakan gambar OS-only dasar atau gambar dasar alternatif, Anda harus menyertakan klien antarmuka runtime dalam gambar Anda. Klien antarmuka runtime memperluasAPI runtime, yang mengelola interaksi antara Lambda dan kode fungsi Anda.

Contoh berikut menunjukkan cara membangun image container untuk.NET menggunakan image AWS non-base, dan cara menambahkan Amazon.Lambda.RuntimeSupport paket, yang merupakan klien antarmuka runtime Lambda untuk.NET. Contoh Dockerfile menggunakan gambar dasar Microsoft .NET 8.

Prasyarat

Untuk menyelesaikan langkah-langkah di bagian ini, Anda harus memiliki hal-hal berikut:

- .NET SDK — Langkah-langkah berikut menggunakan gambar dasar.NET 9. Pastikan versi.NET Anda cocok dengan versi gambar dasar yang Anda tentukan di Dockerfile Anda.
- Docker (versi minimum 25.0.0)
- Plugin buildx Docker.

Membuat dan menyebarkan gambar menggunakan gambar dasar alternatif

1. Instal paket Amazon.Lambda.Templates NuGet .

```
dotnet new install Amazon.Lambda.Templates
```

2. Buat proyek.NET menggunakan `lambda.CustomRuntimeFunction` template. Template ini termasuk Amazon.Lambda.RuntimeSupportpaket.

```
dotnet new lambda.CustomRuntimeFunction --name MyFunction --region us-east-1
```

3. Buka direktori *`MyFunction`*`/src/`*`MyFunction`* tersebut. Di sinilah file proyek disimpan. Periksa file-file berikut:

   - aws-lambda-tools-defaults.json — File ini adalah tempat Anda menentukan opsi baris perintah saat menerapkan fungsi Lambda Anda.
   - Function.cs— Kode berisi kelas dengan `Main` metode yang menginisialisasi `Amazon.Lambda.RuntimeSupport` perpustakaan sebagai bootstrap. `Main`Metode ini adalah titik masuk untuk proses fungsi. `Main`Metode ini membungkus fungsi handler dalam pembungkus yang dapat digunakan bootstrap. Untuk informasi selengkapnya, lihat Menggunakan Amazon.Lambda.RuntimeSupport sebagai pustaka kelas di GitHub repositori.
   - MyFunction.csproj— File proyek.NET, yang mencantumkan file dan rakitan yang terdiri dari aplikasi Anda.
   - Readme.md— File ini berisi informasi lebih lanjut tentang fungsi Lambda sampel.

4. Buka `aws-lambda-tools-defaults.json` file dan Tambahkan baris berikut:

```
  "package-type": "image",
  "docker-host-build-output-dir": "./bin/Release/lambda-publish"
```

- package-type: Mendefinisikan paket deployment sebagai image container.
- docker-host-build-output-dir: Menyetel direktori keluaran untuk proses pembuatan.

Example aws-lambda-alat-defaults.json

```
{
  "Information": [
    "This file provides default values for the deployment wizard inside Visual
 Studio and the AWS Lambda commands added to the .NET Core CLI.",
    "To learn more about the Lambda commands with the .NET Core CLI execute the
 following command at the command line in the project root directory.",
    "dotnet lambda help",
    "All the command line options for the Lambda command can be specified in this
 file."
  ],
  "profile": "",
  "region": "us-east-1",
  "configuration": "Release",
  "function-runtime": "provided.al2023",
  "function-memory-size": 256,
  "function-timeout": 30,
  "function-handler": "bootstrap",
  "msbuild-parameters": "--self-contained true",
  "package-type": "image",
  "docker-host-build-output-dir": "./bin/Release/lambda-publish"
}
```

5. Buat Dockerfile di direktori. *`MyFunction`*`/src/`*`MyFunction`* Contoh berikut Dockerfile menggunakan gambar dasar Microsoft .NET bukan gambar AWS dasar.

- Atur `FROM` properti ke pengenal gambar dasar. Gambar dasar dan `TargetFramework` dalam `MyFunction.csproj` file harus menggunakan versi.NET yang sama.
- Gunakan `COPY` perintah untuk menyalin fungsi ke `/var/task` direktori.

- Atur `ENTRYPOINT` ke modul yang Anda inginkan untuk menjalankan wadah Docker saat dimulai. Dalam hal ini, modul adalah bootstrap, yang menginisialisasi `Amazon.Lambda.RuntimeSupport` perpustakaan.

Perhatikan bahwa contoh Dockerfile tidak menyertakan instruksi USER. Saat Anda menyebarkan gambar kontainer ke Lambda, Lambda secara otomatis mendefinisikan pengguna Linux default dengan izin yang paling tidak memiliki hak istimewa. Ini berbeda dari perilaku Docker standar yang default ke `root` pengguna ketika tidak ada `USER` instruksi yang diberikan.

Example Dockerfile

```
# You can also pull these images from DockerHub amazon/aws-lambda-dotnet:8
FROM mcr.microsoft.com/dotnet/runtime:9.0

# Set the image's internal work directory
WORKDIR /var/task

# Copy function code to Lambda-defined environment variable
COPY "bin/Release/net9.0/linux-x64"  .

# Set the entrypoint to the bootstrap
ENTRYPOINT ["/usr/bin/dotnet", "exec", "/var/task/bootstrap.dll"]
```

6. Instal Amazon.Lambda.Tools ekstensi.NET Global Tools.

```
dotnet tool install -g Amazon.Lambda.Tools
```

Jika Amazon.Lambda.Tools sudah terinstal, pastikan Anda memiliki versi terbaru.

```
dotnet tool update -g Amazon.Lambda.Tools
```

7. Gunakan Amazon.Lambda.Tools untuk membuat image Docker, dorong ke repositori Amazon ECR baru, dan gunakan fungsi Lambda.

Untuk`--function-role`, tentukan nama peran—bukan Amazon Resource Name (ARN) —dari peran eksekusi untuk fungsi tersebut. Misalnya, `lambda-role`.

```
dotnet lambda deploy-function MyFunction --function-role lambda-role
```

Untuk informasi selengkapnya tentang Amazon.Lambda.Tools ekstensi.NET CLI, lihat AWS Ekstensi untuk repositori.NET CLI di. GitHub

8. Memanggil fungsi.

```
dotnet lambda invoke-function MyFunction --payload "Testing the function"
```

Jika semuanya berhasil, Anda akan melihat hal berikut:

```
Payload:
"TESTING THE FUNCTION"

Log Tail:
START RequestId: id Version: $LATEST
END RequestId: id
REPORT RequestId: id  Duration: 0.99 ms       Billed Duration: 1 ms         Memory
 Size: 256 MB     Max Memory Used: 12 MB
```

9. Hapus fungsi Lambda.

```
dotnet lambda delete-function MyFunction
```

Kompilasi kode Fungsi Acara .NET Lambda ke format runtime asli

.NET 8 mendukung kompilasi native ahead-of-time (AOT). Dengan AOT asli, Anda dapat mengkompilasi kode fungsi Lambda Anda ke format runtime asli, yang menghilangkan kebutuhan untuk mengkompilasi kode.NET saat runtime. Kompilasi AOT asli dapat mengurangi waktu mulai dingin untuk fungsi Lambda yang Anda tulis di.NET. Untuk informasi selengkapnya, lihat Memperkenalkan runtime .NET 8 untuk AWS Lambda di AWS Compute Blog.

Bagian-bagian

- waktu aktif Lambda
- Prasyarat
- Memulai
- Serialisasi
- Pemangkasan
- Pemecahan masalah

waktu aktif Lambda

Untuk menerapkan build fungsi Lambda dengan kompilasi AOT asli, gunakan runtime .NET 8 Lambda yang dikelola. Runtime ini mendukung penggunaan arsitektur x86_64 dan arm64.

Saat Anda menerapkan fungsi.NET Lambda tanpa menggunakan AOT, aplikasi Anda pertama kali dikompilasi ke dalam kode Bahasa Menengah (IL). Saat runtime, compiler just-in-time (JIT) di runtime Lambda mengambil kode IL dan mengkompilasinya ke dalam kode mesin sesuai kebutuhan. Dengan fungsi Lambda yang dikompilasi sebelumnya dengan AOT asli, Anda mengkompilasi kode Anda ke dalam kode mesin saat Anda menerapkan fungsi Anda, jadi Anda tidak bergantung pada runtime .NET atau SDK di runtime Lambda untuk mengkompilasi kode Anda sebelum dijalankan.

Salah satu batasan AOT adalah bahwa kode aplikasi Anda harus dikompilasi dalam lingkungan dengan sistem operasi Amazon Linux 2023 (AL2023) yang sama dengan yang digunakan runtime .NET 8. .NET Lambda CLI menyediakan fungsionalitas untuk mengkompilasi aplikasi Anda dalam wadah Docker menggunakan gambar AL2023.

Untuk menghindari potensi masalah dengan kompatibilitas lintas arsitektur, kami sangat menyarankan agar Anda mengkompilasi kode Anda di lingkungan dengan arsitektur prosesor yang sama dengan yang Anda konfigurasikan untuk fungsi Anda. Untuk mempelajari lebih lanjut tentang batasan kompilasi lintas arsitektur, lihat Cross-compilationdi dokumentasi Microsoft .NET.

Prasyarat

Docker

Untuk menggunakan AOT asli, kode fungsi Anda harus dikompilasi dalam lingkungan dengan sistem operasi AL2023 yang sama dengan runtime .NET 8. Perintah.NET CLI di bagian berikut menggunakan Docker untuk mengembangkan dan membangun fungsi Lambda di lingkungan AL2023.

.NET 8 SDK

Kompilasi AOT asli adalah fitur dari .NET 8. Anda harus menginstal .NET 8 SDK pada mesin build Anda, tidak hanya runtime.

Amazon.Lambda.Tools

Untuk membuat fungsi Lambda Anda, Anda menggunakan Amazon.Lambda.Toolsekstensi.NET Global Tools. Untuk menginstalAmazon.Lambda.Tools, jalankan perintah berikut:

```
dotnet tool install -g Amazon.Lambda.Tools
```

Untuk informasi selengkapnya tentang Amazon.Lambda.Tools ekstensi.NET CLI, lihat AWS Ekstensi untuk repositori.NET CLI di. GitHub

Amazon.Lambda.Templates

Untuk menghasilkan kode fungsi Lambda Anda, gunakan paket. Amazon.Lambda.Templates NuGet Untuk menginstal paket template ini, jalankan perintah berikut:

```
dotnet new install Amazon.Lambda.Templates
```

Memulai

Baik .NET Global CLI dan AWS Serverless Application Model (AWS SAM) menyediakan template memulai untuk membangun aplikasi menggunakan AOT asli. Untuk membangun fungsi AOT Lambda asli pertama Anda, lakukan langkah-langkah dalam instruksi berikut.

Untuk menginisialisasi dan menerapkan Fungsi Peristiwa Lambda yang dikompilasi AOT asli

1. Inisialisasi proyek baru menggunakan template AOT asli dan kemudian navigasikan ke direktori yang berisi file yang dibuat `.cs` dan`.csproj`. Dalam contoh ini, kita menamai fungsi kita`NativeAotSample`.

```
dotnet new lambda.NativeAOT -n NativeAotSample
cd ./NativeAotSample/src/NativeAotSample
```

`Function.cs`File yang dibuat oleh template AOT asli berisi kode fungsi berikut.

```
using Amazon.Lambda.Core;
using Amazon.Lambda.RuntimeSupport;
using Amazon.Lambda.Serialization.SystemTextJson;
using System.Text.Json.Serialization;

namespace NativeAotSample;

public class Function
{
    /// <summary>
    /// The main entry point for the Lambda function. The main function is called
 once during the Lambda init phase. It
    /// initializes the .NET Lambda runtime client passing in the function handler
 to invoke for each Lambda event and
    /// the JSON serializer to use for converting Lambda JSON format to the .NET
 types.
    /// </summary>
    private static async Task Main()
    {
        Func<string, ILambdaContext, string> handler = FunctionHandler;
        await LambdaBootstrapBuilder.Create(handler, new
 SourceGeneratorLambdaJsonSerializer<LambdaFunctionJsonSerializerContext>())
            .Build()
            .RunAsync();
    }

    /// <summary>
    /// A simple function that takes a string and does a ToUpper.
    ///
    /// To use this handler to respond to an AWS event, reference the appropriate
 package from
    /// https://github.com/aws/aws-lambda-dotnet#events
    /// and change the string input parameter to the desired event type. When the
 event type
    /// is changed, the handler type registered in the main method needs to be
 updated and the LambdaFunctionJsonSerializerContext
```

```
    /// defined below will need the JsonSerializable updated. If the return type
 and event type are different then the
    /// LambdaFunctionJsonSerializerContext must have two JsonSerializable
 attributes, one for each type.
    ///
    // When using Native AOT extra testing with the deployed Lambda functions is
 required to ensure
    // the libraries used in the Lambda function work correctly with Native AOT. If
 a runtime
    // error occurs about missing types or methods the most likely solution will be
 to remove references to trim-unsafe
    // code or configure trimming options. This sample defaults to partial TrimMode
 because currently the AWS
    // SDK for .NET does not support trimming. This will result in a larger
 executable size, and still does not
    // guarantee runtime trimming errors won't be hit.
    /// </summary>
    /// <param name="input"></param>
    /// <param name="context"></param>
    /// <returns></returns>
    public static string FunctionHandler(string input, ILambdaContext context)
    {
        return input.ToUpper();
    }
}

/// <summary>
/// This class is used to register the input event and return type for the
 FunctionHandler method with the System.Text.Json source generator.
/// There must be a JsonSerializable attribute for each type used as the input and
 return type or a runtime error will occur
/// from the JSON serializer unable to find the serialization information for
 unknown types.
/// </summary>
[JsonSerializable(typeof(string))]
public partial class LambdaFunctionJsonSerializerContext : JsonSerializerContext
{
    // By using this partial class derived from JsonSerializerContext, we can
 generate reflection free JSON Serializer code at compile time
    // which can deserialize our class and properties. However, we must attribute
 this class to tell it what types to generate serialization code for.
    // See https://docs.microsoft.com/en-us/dotnet/standard/serialization/system-
text-json-source-generation
```

Native AOT mengkompilasi aplikasi Anda menjadi satu biner asli. Titik masuk biner itu adalah metodenya. `static Main` Di dalam`static Main`, runtime Lambda di-bootstrap dan metode disiapkan. `FunctionHandler` Sebagai bagian dari bootstrap runtime, serializer yang dihasilkan sumber dikonfigurasi menggunakan `new SourceGeneratorLambdaJsonSerializer<LambdaFunctionJsonSerializerContext>()`

2. Untuk menyebarkan aplikasi Anda ke Lambda, pastikan Docker berjalan di lingkungan lokal Anda dan jalankan perintah berikut.

```
dotnet lambda deploy-function
```

   Di belakang layar, CLI global.NET mengunduh image AL2023 Docker dan mengkompilasi kode aplikasi Anda di dalam wadah yang sedang berjalan. Biner yang dikompilasi adalah output kembali ke sistem file lokal Anda sebelum diterapkan ke Lambda.

3. Uji fungsi Anda dengan menjalankan perintah berikut. Ganti `<FUNCTION_NAME>` dengan nama yang Anda pilih untuk fungsi Anda di wizard penerapan.

```
dotnet lambda invoke-function <FUNCTION_NAME> --payload "hello world"
```

   Respons dari CLI mencakup detail kinerja untuk start dingin (durasi inisialisasi) dan total waktu berjalan untuk pemanggilan fungsi Anda.

4. Untuk menghapus AWS sumber daya yang Anda buat dengan mengikuti langkah-langkah sebelumnya, jalankan perintah berikut. Ganti `<FUNCTION_NAME>` dengan nama yang Anda pilih untuk fungsi Anda di wizard penerapan. Dengan menghapus AWS sumber daya yang tidak lagi Anda gunakan, Anda mencegah tagihan tagihan yang tidak perlu ditagih ke Anda. Akun AWS

```
dotnet lambda delete-function <FUNCTION_NAME>
```

Serialisasi

Untuk menyebarkan fungsi ke Lambda menggunakan AOT asli, kode fungsi Anda harus menggunakan serialisasi yang dihasilkan sumber. Alih-alih menggunakan refleksi run-time untuk mengumpulkan metadata yang diperlukan untuk mengakses properti objek untuk serialisasi, generator sumber menghasilkan file sumber C# yang dikompilasi saat Anda membangun aplikasi Anda. Untuk mengonfigurasi serializer yang dihasilkan sumber Anda dengan benar, pastikan Anda menyertakan objek input dan output apa pun yang digunakan fungsi Anda, serta jenis kustom apa

pun. Misalnya, fungsi Lambda yang menerima peristiwa dari API API Gateway REST API dan mengembalikan `Product` tipe kustom akan menyertakan serializer yang didefinisikan sebagai berikut.

```
[JsonSerializable(typeof(APIGatewayProxyRequest))]
[JsonSerializable(typeof(APIGatewayProxyResponse))]
[JsonSerializable(typeof(Product))]
public partial class CustomSerializer : JsonSerializerContext
{
}
```

Pemangkasan

Native AOT memangkas kode aplikasi Anda sebagai bagian dari kompilasi untuk memastikan bahwa biner sekecil mungkin. .NET 8 untuk Lambda memberikan dukungan pemangkasan yang lebih baik dibandingkan dengan versi sebelumnya .NET. Support telah ditambahkan ke pustaka runtime Lambda, .NET SDK,AWS .NET Lambda Annotations, dan .NET 8 itu sendiri.

Peningkatan ini menawarkan potensi untuk menghilangkan peringatan pemangkasan waktu pembuatan, tetapi .NET tidak pernah sepenuhnya aman. Ini berarti bahwa bagian pustaka yang diandalkan fungsi Anda mungkin dipangkas sebagai bagian dari langkah kompilasi. Anda dapat mengelola ini dengan mendefinisikan `TrimmerRootAssemblies` sebagai bagian dari `.csproj` file Anda seperti yang ditunjukkan pada contoh berikut.

```
<ItemGroup>
    <TrimmerRootAssembly Include="AWSSDK.Core" />
    <TrimmerRootAssembly Include="AWSXRayRecorder.Core" />
    <TrimmerRootAssembly Include="AWSXRayRecorder.Handlers.AwsSdk" />
    <TrimmerRootAssembly Include="Amazon.Lambda.APIGatewayEvents" />
    <TrimmerRootAssembly Include="bootstrap" />
    <TrimmerRootAssembly Include="Shared" />
</ItemGroup>
```

Perhatikan bahwa saat Anda menerima peringatan trim, menambahkan kelas yang menghasilkan peringatan `TrimmerRootAssembly` mungkin tidak menyelesaikan masalah. Peringatan trim menunjukkan bahwa kelas mencoba mengakses beberapa kelas lain yang tidak dapat ditentukan hingga runtime. Untuk menghindari kesalahan runtime, tambahkan kelas kedua ini ke`TrimmerRootAssembly`.

Untuk mempelajari selengkapnya tentang mengelola peringatan trim, lihat Pengantar untuk memangkas peringatan di dokumentasi Microsoft .NET.

Pemecahan masalah

Kesalahan: kompilasi Cross-OS asli tidak didukung.

> Versi alat Amazon.Lambda.Tools global.NET Core Anda sudah ketinggalan zaman. Perbarui ke versi terbaru dan coba lagi.

Docker: sistem operasi gambar “linux” tidak dapat digunakan pada platform ini.

> Docker pada sistem Anda dikonfigurasi untuk menggunakan wadah Windows. Tukar ke wadah Linux untuk menjalankan lingkungan build AOT asli.

Untuk informasi selengkapnya tentang kesalahan umum, lihat repositori AWS NativeAOT for .NET di. GitHub

Menggunakan objek konteks Lambda untuk mengambil informasi Fungsi Acara C

Saat Lambda menjalankan fungsi Anda, Lambda meneruskan objek konteks ke handler. Objek ini menyediakan properti dengan informasi tentang lingkungan invokasi, fungsi, dan eksekusi.

Properti konteks

- `FunctionName` – Nama fungsi Lambda.
- `FunctionVersion` – Versi fungsi.
- `InvokedFunctionArn` – Amazon Resource Name (ARN) yang digunakan untuk memicu fungsi. Menunjukkan jika pemicu menyebutkan nomor versi atau alias.
- `MemoryLimitInMB` – Jumlah memori yang dialokasikan untuk fungsi tersebut.
- `AwsRequestId` – Pengidentifikasi permintaan invokasi.
- `LogGroupName` – Grup log untuk fungsi.
- `LogStreamName` – Aliran log untuk instans fungsi.
- `RemainingTime` (`TimeSpan`) – Jumlah milidetik yang tersisa sebelum waktu eksekusi habis.
- `Identity` – (aplikasi seluler) Informasi tentang identitas Amazon Cognito yang mengesahkan permintaan.
- `ClientContext` – (aplikasi seluler) Konteks klien yang disediakan untuk Lambda oleh aplikasi klien.
- `Logger` Objek logger untuk fungsi.

Anda dapat menggunakan informasi dalam `ILambdaContext` objek untuk menampilkan informasi tentang pemanggilan fungsi Anda untuk tujuan pemantauan. Kode berikut memberikan contoh bagaimana menambahkan informasi konteks ke kerangka logging terstruktur. Dalam contoh ini, fungsi `AwsRequestId` menambah output log. Fungsi ini juga menggunakan `RemainingTime` properti untuk membatalkan tugas dalam penerbangan jika batas waktu fungsi Lambda akan tercapai.

```
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer

namespace GetProductHandler;

public class Function
{
```

```
    private readonly IDatabaseRepository _repo;

    public Function()
    {
        this._repo = new DatabaseRepository();
    }

    public async Task<APIGatewayProxyResponse> FunctionHandler(APIGatewayProxyRequest
 request, ILambdaContext context)
    {
        Logger.AppendKey("AwsRequestId", context.AwsRequestId);

        var id = request.PathParameters["id"];

        using var cts = new CancellationTokenSource();

        try
        {
            cts.CancelAfter(context.RemainingTime.Add(TimeSpan.FromSeconds(-1)));

            var databaseRecord = await this._repo.GetById(id, cts.Token);

            return new APIGatewayProxyResponse
            {
                StatusCode = (int)HttpStatusCode.OK,
                Body = JsonSerializer.Serialize(databaseRecord)
            };
        }
        catch (Exception ex)
        {
            return new APIGatewayProxyResponse
            {
                StatusCode = (int)HttpStatusCode.InternalServerError,
                Body = JsonSerializer.Serialize(new { error = ex.Message })
            };
        }
        finally
        {
            cts.Cancel();
        }
    }
}
```

Log dan pantau Fungsi Acara C# Lambda

AWS Lambda secara otomatis memonitor fungsi Lambda dan mengirim entri log ke Amazon. CloudWatch Fungsi Lambda Anda dilengkapi dengan grup CloudWatch log Log dan aliran log untuk setiap instance fungsi Anda. Lingkungan runtime Lambda mengirimkan detail tentang setiap pemanggilan dan output lainnya dari kode fungsi Anda ke aliran log. Untuk informasi selengkapnya tentang CloudWatch Log, lihatMengirim log Fungsi Peristiwa Lambda ke Log CloudWatch.

Bagian-bagian

- Membuat fungsi yang mengembalikan log
- Menggunakan kontrol logging lanjutan Lambda dengan.NET
- Alat dan pustaka logging tambahan
- Menggunakan Powertools untuk AWS Lambda (.NET) dan AWS SAM untuk logging terstruktur
- Melihat log di konsol Lambda
- Melihat log di CloudWatch konsol
- Melihat log menggunakan AWS Command Line Interface (AWS CLI)
- Menghapus log

Membuat fungsi yang mengembalikan log

Untuk mengeluarkan log dari kode fungsi, Anda dapat menggunakan objek ILambdaLoggerpada konteks, metode di kelas Console, atau pustaka logging apa pun yang menulis ke `stdout` atau`stderr`.

Runtime .NET mencatat baris `START`, `END`, dan `REPORT` untuk setiap invokasi. Baris laporan memberikan perincian berikut.

Laporkan bidang data baris

- RequestId – ID permintaan unik untuk invokasi.
- Durasi – Jumlah waktu yang digunakan oleh metode handler fungsi Anda gunakan untuk memproses peristiwa.
- Durasi yang Ditagih – Jumlah waktu yang ditagihkan untuk invokasi.
- Ukuran Memori – Jumlah memori yang dialokasikan untuk fungsi.
- Memori Maks yang Digunakan – Jumlah memori yang digunakan oleh fungsi. Saat pemanggilan berbagi lingkungan eksekusi, Lambda melaporkan memori maksimum yang digunakan di

semua pemanggilan. Perilaku ini dapat menghasilkan nilai yang dilaporkan lebih tinggi dari yang diharapkan.

- Durasi Init – Untuk permintaan pertama yang dilayani, lama waktu yang diperlukan runtime untuk memuat fungsi dan menjalankan kode di luar metode handler.
- XRAY TraceId — Untuk permintaan yang dilacak, ID AWS X-Ray jejak.
- SegmentId— Untuk permintaan yang dilacak, ID X-Ray segmen.
- Diambil Sampel – Untuk permintaan yang dilacak, hasil pengambilan sampel.

Menggunakan kontrol logging lanjutan Lambda dengan.NET

Untuk memberi Anda kontrol lebih besar atas bagaimana log fungsi Anda ditangkap, diproses, dan digunakan, Anda dapat mengonfigurasi opsi logging berikut untuk runtime .NET yang didukung:

- Format log - pilih antara teks biasa dan format JSON terstruktur untuk log fungsi Anda
- Tingkat log - untuk log dalam format JSON, pilih tingkat detail log yang dikirim CloudWatch Lambda, seperti ERROR, DEBUG, atau INFO
- Grup log - pilih grup CloudWatch log fungsi Anda mengirim log ke

Untuk informasi selengkapnya tentang opsi pencatatan ini, dan petunjuk tentang cara mengonfigurasi fungsi Anda untuk menggunakannya, lihatthe section called “Mengkonfigurasi kontrol logging lanjutan untuk Lambda Event Functions”.

Untuk menggunakan format log dan opsi tingkat log dengan fungsi.NET Lambda Anda, lihat panduan di bagian berikut.

Menggunakan format log JSON terstruktur dengan.NET

Jika Anda memilih JSON untuk format log fungsi Anda, Lambda mengirimkan output log ILambdaLoggermenggunakan JSON terstruktur. Setiap objek log JSON berisi setidaknya lima pasangan nilai kunci dengan kunci berikut:

- `"timestamp"`- waktu pesan log dihasilkan
- `"level"`- tingkat log yang ditetapkan untuk pesan
- `"requestId"`- ID permintaan unik untuk pemanggilan fungsi
- `"traceId"`- variabel `_X_AMZN_TRACE_ID` lingkungan
- `"message"`- isi pesan log

`ILambdaLogger`Instance dapat menambahkan pasangan nilai kunci tambahan, misalnya saat mencatat pengecualian. Anda juga dapat menyediakan parameter tambahan Anda sendiri seperti yang dijelaskan di bagianthe section called “Customer-provided parameter log”.

> **Note**
>
> Jika kode Anda sudah menggunakan pustaka logging lain untuk menghasilkan JSON-formatted log, pastikan format log fungsi Anda disetel ke teks biasa. Menyetel format log ke JSON menghasilkan output log Anda dikodekan ganda.

Contoh perintah logging berikut menunjukkan cara menulis pesan log dengan tingkat`INFO`.

Example Kode logging .NET

```
context.Logger.LogInformation("Fetching cart from database");
```

Anda juga dapat menggunakan metode log generik yang mengambil tingkat log sebagai argumen seperti yang ditunjukkan pada contoh berikut.

```
context.Logger.Log(LogLevel.Information, "Fetching cart from database");
```

Output log oleh cuplikan kode contoh ini akan ditangkap di CloudWatch Log sebagai berikut:

Example Catatan log JSON

```
{
    "timestamp": "2025-09-07T01:30:06.977Z",
    "level": "Information",
    "requestId": "8f711428-7e55-46f9-ae88-2a65d4f85fc5",
    "traceId": "1-6408af34-50f56f5b5677a7d763973804",
    "message": "Fetching cart from database"
}
```

> **Note**
>
> Jika Anda mengonfigurasi format log fungsi Anda untuk menggunakan teks biasa daripada JSON, maka tingkat log yang ditangkap dalam pesan mengikuti konvensi Microsoft menggunakan label empat karakter. Misalnya, tingkat log `Debug` direpresentasikan dalam pesan sebagai`dbug`.

Saat Anda mengonfigurasi fungsi Anda untuk menggunakan log berformat JSON, level log yang ditangkap di log menggunakan label lengkap seperti yang ditunjukkan pada contoh catatan log JSON.

Jika Anda tidak menetapkan level ke output log Anda, Lambda secara otomatis menetapkannya INFO level.

Pengecualian logging di JSON

Saat menggunakan logging JSON terstrukturILambdaLogger, Anda dapat mencatat pengecualian dalam kode Anda seperti yang ditunjukkan pada contoh berikut.

Example penggunaan log pengecualian

```
try
{
    connection.ExecuteQuery(query);
}
catch(Exception e)
{
    context.Logger.LogWarning(e, "Error executing query");
}
```

Output format log oleh kode ini ditunjukkan dalam contoh berikut JSON. Perhatikan bahwa `message` properti di JSON diisi menggunakan argumen pesan yang disediakan dalam `LogWarning` panggilan, sedangkan `errorMessage` properti berasal dari `Message` properti pengecualian itu sendiri.

Example Catatan log JSON

```
{
    "timestamp": "2025-09-07T01:30:06.977Z",
    "level": "Warning",
    "requestId": "8f711428-7e55-46f9-ae88-2a65d4f85fc5",
    "traceId": "1-6408af34-50f56f5b5677a7d763973804",
    "message": "Error executing query",
    "errorType": "System.Data.SqlClient.SqlException",
    "errorMessage": "Connection closed",
    "stackTrace": ["<call exception.StackTrace>"]
}
```

Jika format logging fungsi Anda disetel ke JSON, Lambda juga JSON-formatted mengeluarkan pesan log saat kode Anda melempar pengecualian yang tidak tertangkap. Contoh cuplikan kode berikut dan pesan log menunjukkan bagaimana pengecualian yang tidak tertangkap dicatat.

Example kode pengecualian

```
throw new ApplicationException("Invalid data");
```

Example Catatan log JSON

```
{
    "timestamp": "2025-09-07T01:30:06.977Z",
    "level": "Error",
    "requestId": "8f711428-7e55-46f9-ae88-2a65d4f85fc5",
    "traceId": "1-6408af34-50f56f5b5677a7d763973804",
    "message": "Invalid data",
    "errorType": "System.ApplicationException",
    "errorMessage": "Invalid data",
    "stackTrace": ["<call exception.StackTrace>"]
}
```

Customer-provided parameter log

Dengan pesan JSON-formatted log, Anda dapat menyediakan parameter log tambahan dan memasukkannya ke dalam log`message`. Contoh cuplikan kode berikut menunjukkan perintah untuk menambahkan dua parameter yang disediakan pengguna berlabel dan. `retryAttempt` `uri` Dalam contoh, nilai parameter ini berasal dari `uriDestination` argumen `retryAttempt` dan diteruskan ke perintah logging.

Example Perintah logging JSON dengan parameter tambahan

```
context.Logger.LogInformation("Starting retry {retryAttempt} to make GET request to
 {uri}", retryAttempt, uriDestination);
```

Output pesan log oleh perintah ini ditunjukkan dalam contoh berikut JSON.

Example Catatan log JSON

```
{
    "timestamp": "2025-09-07T01:30:06.977Z",
```

```
    "level": "Information",
    "requestId": "8f711428-7e55-46f9-ae88-2a65d4f85fc5",
    "traceId": "1-6408af34-50f56f5b5677a7d763973804",
    "message": "Starting retry 1 to make GET request to http://example.com/",
    "retryAttempt": 1,
    "uri": "http://example.com/"
}
```

Tip

Anda juga dapat menggunakan properti posisi alih-alih nama saat menentukan parameter tambahan. Misalnya, perintah logging pada contoh sebelumnya juga dapat ditulis sebagai berikut:

```
context.Logger.LogInformation("Starting retry {0} to make GET request to {1}",
 retryAttempt, uriDestination);
```

Perhatikan bahwa saat Anda menyediakan parameter logging tambahan, Lambda menangkapnya sebagai properti tingkat atas dalam catatan log JSON. Pendekatan ini berbeda dari beberapa pustaka logging .NET populer seperti`Serilog`, yang menangkap parameter tambahan dalam objek anak yang terpisah.

Jika argumen yang Anda berikan untuk parameter tambahan adalah objek yang kompleks, secara default Lambda menggunakan `ToString()` metode untuk memberikan nilai. Untuk menunjukkan bahwa argumen harus JSON serial, gunakan `@` awalan seperti yang ditunjukkan dalam cuplikan kode berikut. Dalam contoh ini, `User` adalah objek dengan `FirstName` dan `LastName` properti.

Example Perintah logging JSON dengan objek serial JSON

```
context.Logger.LogInformation("User {@user} logged in", User);
```

Output pesan log oleh perintah ini ditunjukkan dalam contoh berikut JSON.

Example Catatan log JSON

```
{
    "timestamp": "2025-09-07T01:30:06.977Z",
    "level": "Information",
```

```
    "requestId": "8f711428-7e55-46f9-ae88-2a65d4f85fc5",
    "traceId": "1-6408af34-50f56f5b5677a7d763973804",
    "message": "User {@user} logged in",
    "user":
    {
        "FirstName": "John",
        "LastName": "Doe"
    }
}
```

Jika argumen untuk parameter tambahan adalah array atau mengimplementasikan `IList` atau`IDictionary`, maka Lambda menambahkan argumen ke pesan log JSON sebagai array seperti yang ditunjukkan dalam contoh berikut catatan log JSON. Dalam contoh ini, `{users}` mengambil `IList` argumen yang berisi instance `User` properti dengan format yang sama seperti contoh sebelumnya. Lambda mengubah ini `IList` menjadi array, dengan setiap nilai yang dibuat menggunakan metode. `ToString`

Example Catatan log JSON dengan argumen **`IList`**

```
{
    "timestamp": "2025-09-07T01:30:06.977Z",
    "level": "Information",
    "requestId": "8f711428-7e55-46f9-ae88-2a65d4f85fc5",
    "traceId": "1-6408af34-50f56f5b5677a7d763973804",
    "message": "{users} have joined the group",
    "users":
    [
        "Rosalez, Alejandro",
        "Stiles, John"
    ]
}
```

Anda juga dapat membuat serial JSON daftar dengan menggunakan `@` awalan dalam perintah logging Anda. Dalam contoh catatan log JSON berikut, `users` properti adalah serial JSON.

Example **Catatan log JSON dengan argumen iList serial JSON**

```
{
    "timestamp": "2025-09-07T01:30:06.977Z",
    "level": "Information",
    "requestId": "8f711428-7e55-46f9-ae88-2a65d4f85fc5",
    "traceId": "1-6408af34-50f56f5b5677a7d763973804",
```

```
    "message": "{@users} have joined the group",
    "users":
    [
        {
            "FirstName": "Alejandro",
            "LastName": "Rosalez"
        },
        {
            "FirstName": "John",
            "LastName": "Stiles"
        }
    ]
}
```

Menggunakan penyaringan tingkat log dengan.NET

Dengan mengonfigurasi penyaringan tingkat log, Anda dapat memilih untuk mengirim hanya log dengan tingkat detail tertentu atau lebih rendah ke Log. CloudWatch Untuk mempelajari cara mengonfigurasi pemfilteran tingkat log untuk fungsi Anda, lihat. the section called “Log-level penyaringan”

AWS Lambda Untuk memfilter pesan log Anda berdasarkan tingkat log, Anda dapat menggunakan log berformat JSON atau menggunakan `Console` metode.NET untuk menampilkan pesan log. Untuk membuat log berformat JSON, konfigurasikan jenis log fungsi Anda ke JSON dan gunakan instance. `ILambdaLogger`

Dengan JSON-formatted log, Lambda memfilter output log Anda menggunakan pasangan nilai kunci “level” di objek JSON yang dijelaskan di. the section called “Menggunakan format log JSON terstruktur dengan.NET”

Jika Anda menggunakan `Console` metode.NET untuk menulis pesan ke CloudWatch Log, Lambda menerapkan tingkat log ke pesan Anda sebagai berikut:

- Console.WriteLine metode - Lambda menerapkan log-level `INFO`
- Console.Error metode - Lambda menerapkan log-level `ERROR`

Ketika Anda mengonfigurasi fungsi Anda untuk menggunakan pemfilteran tingkat log, Anda harus memilih dari opsi berikut untuk tingkat log yang ingin Lambda kirim ke Log. CloudWatch Perhatikan pemetaan level log yang digunakan oleh Lambda dengan level Microsoft standar yang digunakan oleh.NET. `ILambdaLogger`

Tingkat log Lambda	Tingkat Microsoft yang setara	Penggunaan standar
TRACE (paling detail)	Jejak	Informasi paling halus yang digunakan untuk melacak jalur eksekusi kode Anda
DEBUG	Debug	Informasi terperinci untuk debugging sistem
INFO	Informasi	Pesan yang merekam operasi normal fungsi Anda
WARN	Peringatan	Pesan tentang potensi kesalahan yang dapat menyebabkan perilaku tak terduga jika tidak ditangani
Kesalahan	Kesalahan	Pesan tentang masalah yang mencegah kode berfungsi seperti yang diharapkan
FATAL (paling detail)	Kritis	Pesan tentang kesalahan serius yang menyebabkan aplikasi berhenti berfungsi

Lambda mengirimkan log dari tingkat detail yang dipilih dan lebih rendah ke. CloudWatch Misalnya, jika Anda mengonfigurasi level log WARN, Lambda mengirimkan log yang sesuai dengan level WARN, ERROR, dan FATAL.

Alat dan pustaka logging tambahan

Powertools for AWS Lambda (.NET) adalah toolkit pengembang untuk mengimplementasikan praktik terbaik Tanpa Server dan meningkatkan kecepatan pengembang. Utilitas Logging menyediakan logger yang dioptimalkan Lambda yang mencakup informasi tambahan tentang konteks fungsi di semua fungsi Anda dengan output terstruktur sebagai JSON. Gunakan utilitas ini untuk melakukan hal berikut:

- Tangkap bidang kunci dari konteks Lambda, cold start, dan struktur logging output sebagai JSON

- Log peristiwa pemanggilan Lambda saat diinstruksikan (dinonaktifkan secara default)
- Cetak semua log hanya untuk persentase pemanggilan melalui pengambilan sampel log (dinonaktifkan secara default)
- Tambahkan kunci tambahan ke log terstruktur kapan saja
- Gunakan pemformat log kustom (Bring Your Own Formatter) untuk mengeluarkan log dalam struktur yang kompatibel dengan RFC Logging organisasi Anda

Menggunakan Powertools untuk AWS Lambda (.NET) dan AWS SAM untuk logging terstruktur

Ikuti langkah-langkah di bawah ini untuk mengunduh, membangun, dan menyebarkan contoh aplikasi Hello World C # dengan Powertools terintegrasi untuk modul AWS Lambda (.NET) menggunakan modul. AWS SAM Aplikasi ini mengimplementasikan backend API dasar dan menggunakan Powertools untuk memancarkan log, metrik, dan jejak. Ini terdiri dari titik akhir Amazon API Gateway dan fungsi Lambda. Saat Anda mengirim permintaan GET ke titik akhir API Gateway, fungsi Lambda memanggil, mengirim log dan metrik menggunakan Format Metrik Tertanam CloudWatch ke, dan mengirimkan jejak ke. AWS X-Ray Fungsi mengembalikan `hello world` pesan.

Prasyarat

Untuk menyelesaikan langkah-langkah di bagian ini, Anda harus memiliki hal-hal berikut:

- .NET 8
- AWS CLI versi 2
- AWS SAM CLI versi 1.75 atau yang lebih baru. Jika Anda memiliki versi CLI yang lebih lama, lihat Memutakhirkan AWS SAM CLI. AWS SAM

Menyebarkan sampel AWS SAM application

1. Inisialisasi aplikasi menggunakan TypeScript template Hello World.

```
sam init --app-template hello-world-powertools-dotnet --name sam-app --package-type
 Zip --runtime dotnet6 --no-tracing
```

2. Bangun aplikasi.

```
cd sam-app && sam build
```

3. Terapkan aplikasi.

```
sam deploy --guided
```

4. Ikuti petunjuk di layar. Untuk menerima opsi default yang disediakan dalam pengalaman interaktif, tekan`Enter`.

> **Note**
>
> Karena HelloWorldFunction mungkin tidak memiliki otorisasi yang ditentukan, Apakah ini baik-baik saja? , pastikan untuk masuky.

5. Dapatkan URL aplikasi yang digunakan:

```
aws cloudformation describe-stacks --stack-name sam-app --query
 'Stacks[0].Outputs[?OutputKey==`HelloWorldApi`].OutputValue' --output text
```

6. Memanggil titik akhir API:

```
curl -X GET <URL_FROM_PREVIOUS_STEP>
```

Jika berhasil, Anda akan melihat tanggapan ini:

```
{"message":"hello world"}
```

7. Untuk mendapatkan log untuk fungsi tersebut, jalankan log sam. Untuk informasi selengkapnya, lihat Bekerja dengan log di Panduan AWS Serverless Application Model Pengembang.

```
sam logs --stack-name sam-app
```

Output log terlihat seperti ini:

```
2025/02/20/[$LATEST]4eaf8445ba7a4a93b999cb17fbfbecd8
 2025-09-20T14:15:27.988000 INIT_START Runtime Version:
 dotnet:6.v13        Runtime Version ARN: arn:aws:lambda:ap-
southeast-2::runtime:699f346a05dae24c58c45790bc4089f252bf17dae3997e79b17d939a288aa1ec
2025/02/20/[$LATEST]4eaf8445ba7a4a93b999cb17fbfbecd8 2025-09-20T14:15:28.229000
 START RequestId: bed25b38-d012-42e7-ba28-f272535fb80e Version: $LATEST
2025/02/20/[$LATEST]4eaf8445ba7a4a93b999cb17fbfbecd8 2025-09-20T14:15:29.259000
 2025-09-20T14:15:29.201Z        bed25b38-d012-42e7-ba28-f272535fb80e    info
   {"_aws":{"Timestamp":1676902528962,"CloudWatchMetrics":[{"Namespace":"sam-
```

```
app-logging","Metrics":[{"Name":"ColdStart","Unit":"Count"}],"Dimensions":
[["FunctionName"],["Service"]]}]},"FunctionName":"sam-app-HelloWorldFunction-
haKIoVeose2p","Service":"PowertoolsHelloWorld","ColdStart":1}
2025/02/20/[$LATEST]4eaf8445ba7a4a93b999cb17fbfbecd8 2025-09-20T14:15:30.479000
 2025-09-20T14:15:30.479Z        bed25b38-d012-42e7-ba28-f272535fb80e    info
 {"ColdStart":true,"XrayTraceId":"1-63f3807f-5dbcb9910c96f50742707542","CorrelationId":"d3d
a549-4d67b2fdc015","FunctionName":"sam-app-HelloWorldFunction-
haKIoVeose2p","FunctionVersion":"$LATEST","FunctionMemorySize":256,"FunctionArn":"arn:aws:l
southeast-2:123456789012:function:sam-app-HelloWorldFunction-
haKIoVeose2p","FunctionRequestId":"bed25b38-d012-42e7-ba28-
f272535fb80e","Timestamp":"2025-09-20T14:15:30.4602970Z","Level":"Information","Service":"P
 world API - HTTP 200"}
2025/02/20/[$LATEST]4eaf8445ba7a4a93b999cb17fbfbecd8 2025-09-20T14:15:30.599000
 2025-09-20T14:15:30.599Z        bed25b38-d012-42e7-ba28-f272535fb80e    info
   {"_aws":{"Timestamp":1676902528922,"CloudWatchMetrics":[{"Namespace":"sam-
app-logging","Metrics":[{"Name":"ApiRequestCount","Unit":"Count"}],"Dimensions":
[["Service"]]}]},"Service":"PowertoolsHelloWorld","ApiRequestCount":1}
2025/02/20/[$LATEST]4eaf8445ba7a4a93b999cb17fbfbecd8 2025-09-20T14:15:30.680000 END
 RequestId: bed25b38-d012-42e7-ba28-f272535fb80e
2025/02/20/[$LATEST]4eaf8445ba7a4a93b999cb17fbfbecd8 2025-09-20T14:15:30.680000
 REPORT RequestId: bed25b38-d012-42e7-ba28-f272535fb80e  Duration: 2450.99 ms
  Billed Duration: 2692 ms Memory Size: 256 MB     Max Memory Used: 74 MB  Init
 Duration: 240.05 ms
XRAY TraceId: 1-63f3807f-5dbcb9910c96f50742707542       SegmentId: 16b362cd5f52cba0
```

8. Ini adalah titik akhir API publik yang dapat diakses melalui internet. Kami menyarankan Anda menghapus titik akhir setelah pengujian.

```
sam delete
```

Mengelola retensi log

Grup log tidak terhapus secara otomatis ketika Anda menghapus suatu fungsi. Untuk menghindari penyimpanan log tanpa batas waktu, hapus grup log, atau konfigurasikan periode retensi setelah itu secara CloudWatch otomatis menghapus log. Untuk mengatur penyimpanan log, tambahkan yang berikut ini ke AWS SAM templat Anda:

```
Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function
    Properties:
      # Omitting other properties
```

```
  LogGroup:
    Type: AWS::Logs::LogGroup
    Properties:
      LogGroupName: !Sub "/aws/lambda/${HelloWorldFunction}"
      RetentionInDays: 7
```

Melihat log di konsol Lambda

Anda dapat menggunakan konsol Lambda untuk melihat keluaran log setelah Anda menjalankan Fungsi Peristiwa Lambda.

Jika kode Anda dapat diuji dari editor Kode tertanam, Anda menemukan log dalam hasil eksekusi. Saat Anda menggunakan fitur pengujian konsol untuk menjalankan fungsi, Anda menemukan Keluaran Log di bagian Detail.

Melihat log di CloudWatch konsol

Anda dapat menggunakan CloudWatch konsol Amazon untuk melihat log untuk semua pemanggilan Fungsi Peristiwa Lambda.

Untuk melihat log di CloudWatch konsol

1. Buka halaman Grup log di CloudWatch konsol.
2. Pilih grup log untuk fungsi Anda (/aws/lambda/***your-function-name***).
3. Pilih pengaliran log.

Setiap aliran log sesuai dengan instans fungsi Anda. Aliran log muncul saat Anda memperbarui fungsi Lambda, dan saat instance tambahan dibuat untuk menangani pemanggilan bersamaan. Untuk menemukan log untuk pemanggilan tertentu, kami sarankan untuk menginstrumentasi fungsi Anda dengan. AWS X-Ray X-Ray mencatat detail tentang permintaan dan aliran log dalam jejak.

Melihat log menggunakan AWS Command Line Interface (AWS CLI)

AWS CLI Ini adalah alat sumber terbuka yang dapat Anda gunakan untuk berinteraksi dengan AWS layanan menggunakan perintah di shell baris perintah Anda. Untuk menyelesaikan langkah-langkah di bagian ini, Anda harus memiliki AWS CLI versi 2.

Anda dapat menggunakan AWS CLI untuk mengambil log untuk invokasi menggunakan opsi perintah `--log-type`. Respons berisi bidang `LogResult` yang memuat hingga 4 KB log berkode base64 dari invokasi.

Example mengambil ID log

Contoh berikut menunjukkan cara mengambil ID log dari `LogResult` untuk fungsi bernama `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

Anda akan melihat output berikut:

```
{
    "StatusCode": 200,
    "LogResult":
 "U1RBUlQgUmVxdWVzdElkOiA4N2QwNDRiOC1mMTU0LTExZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "ExecutedVersion": "$LATEST"
}
```

Example mendekode log

Pada prompt perintah yang sama, gunakan utilitas `base64` untuk mendekodekan log. Contoh berikut menunjukkan cara mengambil log berkode base64 untuk `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail \
--query 'LogResult' --output text --cli-binary-format raw-in-base64-out | base64 --
decode
```

cli-binary-formatOpsi ini diperlukan jika Anda menggunakan AWS CLI versi 2. Untuk menjadikan ini pengaturan default, jalankan`aws configure set cli-binary-format raw-in-base64-out`. Untuk informasi selengkapnya, lihat opsi baris perintah global yang AWS CLI didukung di Panduan AWS Command Line Interface Pengguna untuk Versi 2.

Anda akan melihat output berikut:

```
START RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Version: $LATEST
"AWS_SESSION_TOKEN": "AgoJb3JpZ2luX2VjELj...", "_X_AMZN_TRACE_ID": "Root=1-5d02e5ca-
f5792818b6fe8368e5b51d50;Parent=191db58857df8395;Sampled=0"",ask/lib:/opt/lib",
END RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8
REPORT RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8  Duration: 79.67 ms      Billed
 Duration: 80 ms         Memory Size: 128 MB     Max Memory Used: 73 MB
```

Utilitas `base64` tersedia di Linux, macOS, dan Ubuntu pada Windows. Pengguna macOS mungkin harus menggunakan `base64 -D`.

Example Skrip get-logs.sh

Pada prompt perintah yang sama, gunakan script berikut untuk mengunduh lima peristiwa log terakhir. Skrip menggunakan `sed` untuk menghapus kutipan dari file output, dan akan tidur selama 15 detik untuk memberikan waktu agar log tersedia. Output mencakup respons dari Lambda dan output dari perintah `get-log-events`.

Salin konten dari contoh kode berikut dan simpan dalam direktori proyek Lambda Anda sebagai `get-logs.sh`.

cli-binary-formatOpsi ini diperlukan jika Anda menggunakan AWS CLI versi 2. Untuk menjadikan ini pengaturan default, jalankan`aws configure set cli-binary-format raw-in-base64-out`. Untuk informasi selengkapnya, lihat opsi baris perintah global yang AWS CLI didukung di Panduan AWS Command Line Interface Pengguna untuk Versi 2.

```
#!/bin/bash
aws lambda invoke --function-name my-function --cli-binary-format raw-in-base64-out --
payload '{"key": "value"}' out
sed -i'' -e 's/"//g' out
sleep 15
aws logs get-log-events --log-group-name /aws/lambda/my-function --log-stream-
name stream1 --limit 5
```

Example macOS dan Linux (khusus)

Dalam prompt perintah yang sama, pengguna macOS dan Linux mungkin perlu menjalankan perintah berikut untuk memastikan skrip dapat dieksekusi.

```
chmod -R 755 get-logs.sh
```

Example mengambil lima log acara terakhir

Pada prompt perintah yang sama, gunakan skrip berikut untuk mendapatkan lima log acara terakhir.

```
./get-logs.sh
```

Anda akan melihat output berikut:

```
{
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
```

```
}
{
    "events": [
        {
            "timestamp": 1559763003171,
            "message": "START RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf Version:
 $LATEST\n",
            "ingestionTime": 1559763003309
        },
        {
            "timestamp": 1559763003173,
            "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tENVIRONMENT VARIABLES\r{\r  \"AWS_LAMBDA_FUNCTION_VERSION\": \"$LATEST\",
\r ...",
            "ingestionTime": 1559763018353
        },
        {
            "timestamp": 1559763003173,
            "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tEVENT\r{\r  \"key\": \"value\"\r}\n",
            "ingestionTime": 1559763018353
        },
        {
            "timestamp": 1559763003218,
            "message": "END RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf\n",
            "ingestionTime": 1559763018353
        },
        {
            "timestamp": 1559763003218,
            "message": "REPORT RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tDuration: 26.73 ms\tBilled Duration: 27 ms \tMemory Size: 128 MB\tMax Memory Used: 75
 MB\t\n",
            "ingestionTime": 1559763018353
        }
    ],
    "nextForwardToken": "f/34783877304859518393868359594929986069206639495374241795",
    "nextBackwardToken": "b/34783877303811383369537420289090800615709599058929582080"
}
```

Menghapus log

Grup log tidak terhapus secara otomatis ketika Anda menghapus suatu fungsi. Untuk menghindari penyimpanan log secara tidak terbatas, hapus kelompok log, atau lakukan konfigurasi periode penyimpanan, yang setelahnya log akan dihapus secara otomatis.

Menginstrumentasi kode C # di AWS Lambda

Lambda terintegrasi dengan AWS X-Ray untuk membantu Anda melacak, men-debug, dan mengoptimalkan aplikasi Lambda. Anda dapat menggunakan X-Ray untuk melacak permintaan saat melintasi sumber daya dalam aplikasi Anda, yang mungkin termasuk fungsi Lambda dan layanan lainnya. AWS

Untuk mengirim data penelusuran X-Ray, Anda dapat menggunakan salah satu dari tiga pustaka SDK:

- AWS Distro for OpenTelemetry (ADOT) — Distribusi SDK (OTel) yang aman, siap produksi, dan AWS didukung. OpenTelemetry
- AWS X-Ray SDK for .NET— SDK untuk menghasilkan dan mengirim data jejak ke X-Ray.
- Powertools for AWS Lambda (.NET) — Toolkit pengembang untuk menerapkan praktik terbaik Tanpa Server dan meningkatkan kecepatan pengembang.

Setiap SDK menawarkan cara untuk mengirim data telemetri Anda ke layanan. X-Ray Anda kemudian dapat menggunakan X-Ray untuk melihat, memfilter, dan mendapatkan wawasan tentang metrik kinerja aplikasi Anda untuk mengidentifikasi masalah dan peluang untuk pengoptimalan.

> **Important**
>
> The X-Ray dan Powertools untuk AWS Lambda SDK adalah bagian dari solusi instrumentasi terintegrasi ketat yang ditawarkan oleh. AWS Lapisan Lambda ADOT adalah bagian dari standar industri untuk melacak instrumentasi yang mengumpulkan lebih banyak data secara umum, tetapi mungkin tidak cocok untuk semua kasus penggunaan. Anda dapat menerapkan penelusuran ujung ke ujung dalam X-Ray menggunakan salah satu solusi. Untuk mempelajari lebih lanjut tentang memilih di antara keduanya, lihat Memilih antara AWS Distro untuk Telemetri Terbuka dan SDK. X-Ray

Bagian-bagian

- Menggunakan Powertools untuk AWS Lambda (.NET) dan AWS SAM untuk menelusuri
- Menggunakan X-Ray SDK untuk instrumen fungsi.NET Anda
- Mengaktifkan penelusuran dengan konsol Lambda
- Mengaktifkan penelusuran dengan Lambda API

- Mengaktifkan penelusuran dengan CloudFormation
- Menafsirkan jejak X-Ray

Menggunakan Powertools untuk AWS Lambda (.NET) dan AWS SAM untuk menelusuri

Ikuti langkah-langkah di bawah ini untuk mengunduh, membangun, dan menyebarkan contoh aplikasi Hello World C # dengan Powertools terintegrasi untuk modul AWS Lambda (.NET) menggunakan modul. AWS SAM Aplikasi ini mengimplementasikan backend API dasar dan menggunakan Powertools untuk memancarkan log, metrik, dan jejak. Ini terdiri dari titik akhir Amazon API Gateway dan fungsi Lambda. Saat Anda mengirim permintaan GET ke titik akhir API Gateway, fungsi Lambda memanggil, mengirim log dan metrik menggunakan Format Metrik Tertanam CloudWatch ke, dan mengirimkan jejak ke. AWS X-Ray Fungsi mengembalikan pesan hello world.

Prasyarat

Untuk menyelesaikan langkah-langkah di bagian ini, Anda harus memiliki hal-hal berikut:

- .NET 8
- AWS CLI versi 2
- AWS SAM CLI versi 1.75 atau yang lebih baru. Jika Anda memiliki versi CLI yang lebih lama, lihat Memutakhirkan AWS SAM CLI. AWS SAM

Menyebarkan sampel AWS SAM application

1. Inisialisasi aplikasi menggunakan TypeScript template Hello World.

```
sam init --app-template hello-world-powertools-dotnet --name sam-app --package-type
 Zip --runtime dotnet6 --no-tracing
```

2. Bangun aplikasi.

```
cd sam-app && sam build
```

3. Terapkan aplikasi.

```
sam deploy --guided
```

4. Ikuti petunjuk di layar. Untuk menerima opsi default yang disediakan dalam pengalaman interaktif, tekan`Enter`.

> **Note**
>
> Karena HelloWorldFunction mungkin tidak memiliki otorisasi yang ditentukan, Apakah ini baik-baik saja? , pastikan untuk masuky.

5. Dapatkan URL aplikasi yang digunakan:

```
aws cloudformation describe-stacks --stack-name sam-app --query
 'Stacks[0].Outputs[?OutputKey==`HelloWorldApi`].OutputValue' --output text
```

6. Memanggil titik akhir API:

```
curl <URL_FROM_PREVIOUS_STEP>
```

Jika berhasil, Anda akan melihat tanggapan ini:

```
{"message":"hello world"}
```

7. Untuk mendapatkan jejak untuk fungsi tersebut, jalankan jejak sam.

```
sam traces
```

Output jejak terlihat seperti ini:

```
New XRay Service Graph
  Start time: 2023-02-20 23:05:16+08:00
  End time: 2023-02-20 23:05:16+08:00
  Reference Id: 0 - AWS::Lambda - sam-app-HelloWorldFunction-pNjujb7mEoew - Edges:
 [1]
   Summary_statistics:
     - total requests: 1
     - ok count(2XX): 1
     - error count(4XX): 0
     - fault count(5XX): 0
     - total response time: 2.814
  Reference Id: 1 - AWS::Lambda::Function - sam-app-HelloWorldFunction-pNjujb7mEoew
 - Edges: []
   Summary_statistics:
     - total requests: 1
     - ok count(2XX): 1
```

```
    - error count(4XX): 0
    - fault count(5XX): 0
    - total response time: 2.429
  Reference Id: 2 - (Root) AWS::ApiGateway::Stage - sam-app/Prod - Edges: [0]
   Summary_statistics:
    - total requests: 1
    - ok count(2XX): 1
    - error count(4XX): 0
    - fault count(5XX): 0
    - total response time: 2.839
  Reference Id: 3 - client - sam-app/Prod - Edges: [2]
   Summary_statistics:
    - total requests: 0
    - ok count(2XX): 0
    - error count(4XX): 0
    - fault count(5XX): 0
    - total response time: 0

XRay Event [revision 3] at (2023-02-20T23:05:16.521000) with id
 (1-63f38c2c-270200bf1d292a442c8e8a00) and duration (2.877s)
 - 2.839s - sam-app/Prod [HTTP: 200]
   - 2.836s - Lambda [HTTP: 200]
 - 2.814s - sam-app-HelloWorldFunction-pNjujb7mEoew [HTTP: 200]
 - 2.429s - sam-app-HelloWorldFunction-pNjujb7mEoew
   - 0.230s - Initialization
   - 2.389s - Invocation
     - 0.600s - ## FunctionHandler
       - 0.517s - Get Calling IP
   - 0.039s - Overhead
```

8. Ini adalah titik akhir API publik yang dapat diakses melalui internet. Kami menyarankan Anda menghapus titik akhir setelah pengujian.

```
sam delete
```

X-Ray tidak melacak semua permintaan ke aplikasi Anda. X-Ray menerapkan algoritma pengambilan sampel untuk memastikan bahwa penelusuran efisien, sambil tetap memberikan sampel yang representatif dari semua permintaan. Tingkat pengambilan sampel adalah 1 permintaan per detik dan 5 persen dari permintaan tambahan. Anda tidak dapat mengonfigurasi laju X-Ray pengambilan sampel untuk fungsi Anda.

Menggunakan X-Ray SDK untuk instrumen fungsi.NET Anda

Anda dapat menggunakan instrumen kode fungsi Anda untuk merekam metadata dan melacak panggilan downstream. Untuk merekam detail tentang panggilan yang dilakukan fungsi Anda ke sumber daya dan layanan lain, gunakan file AWS X-Ray SDK for .NET. Untuk mendapatkan SDK, tambahkan paket `AWSXRayRecorder` ke file proyek Anda.

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <TargetFramework>net8.0</TargetFramework>
    <GenerateRuntimeConfigurationFiles>true</GenerateRuntimeConfigurationFiles>
    <AWSProjectType>Lambda</AWSProjectType>
  </PropertyGroup>
  <ItemGroup>
    <PackageReference Include="Amazon.Lambda.Core" Version="2.1.0" />
    <PackageReference Include="Amazon.Lambda.SQSEvents" Version="2.1.0" />
    <PackageReference Include="Amazon.Lambda.Serialization.Json" Version="2.1.0" />
    <PackageReference Include="AWSSDK.Core" Version="3.7.103.24" />
    <PackageReference Include="AWSSDK.Lambda" Version="3.7.104.3" />
    <PackageReference Include="AWSXRayRecorder.Core" Version="2.13.0" />
    <PackageReference Include="AWSXRayRecorder.Handlers.AwsSdk" Version="2.11.0" />
  </ItemGroup>
</Project>
```

Ada berbagai paket Nuget yang menyediakan instrumentasi otomatis untuk AWS SDK, Entity Framework, dan permintaan HTTP. Untuk melihat set lengkap opsi konfigurasi, lihat AWS X-Ray SDK for .NET di Panduan AWS X-Ray Pengembang.

Setelah Anda menambahkan paket Nuget yang diinginkan, konfigurasikan instrumentasi otomatis. Praktik terbaik adalah melakukan konfigurasi ini di luar fungsi handler fungsi Anda. Ini memanfaatkan penggunaan kembali lingkungan eksekusi untuk meningkatkan kinerja fungsi Anda. Dalam contoh kode berikut, `RegisterXRayForAllServices` metode ini dipanggil dalam konstruktor fungsi untuk menambahkan instrumentasi untuk semua panggilan AWS SDK.

```
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer

namespace GetProductHandler;

public class Function
{
```

```
    private readonly IDatabaseRepository _repo;

    public Function()
    {
        // Add auto instrumentation for all AWS SDK calls
        // It is important to call this method before initializing any SDK clients
        AWSSDKHandler.RegisterXRayForAllServices();
        this._repo = new DatabaseRepository();
    }

    public async Task<APIGatewayProxyResponse> FunctionHandler(APIGatewayProxyRequest
 request)
    {
        var id = request.PathParameters["id"];

        var databaseRecord = await this._repo.GetById(id);

        return new APIGatewayProxyResponse
        {
            StatusCode = (int)HttpStatusCode.OK,
            Body = JsonSerializer.Serialize(databaseRecord)
        };
    }
}
```

Mengaktifkan penelusuran dengan konsol Lambda

Untuk mengaktifkan penelusuran aktif di Fungsi Peristiwa Lambda Anda dengan konsol, ikuti langkah-langkah berikut:

Untuk mengaktifkan penelusuran aktif

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih Fungsi Acara.
3. Pilih Konfigurasi dan kemudian pilih Alat Pemantauan dan operasi.
4. Di bawah Alat pemantauan tambahan, pilih Edit.
5. Di bawah Sinyal CloudWatch Aplikasi dan AWS X-Ray, pilih Aktifkan untuk jejak layanan Lambda.
6. Pilih Simpan.

Mengaktifkan penelusuran dengan Lambda API

Konfigurasikan penelusuran pada fungsi Lambda Anda dengan AWS CLI AWS atau SDK, gunakan operasi API berikut:

- UpdateFunctionConfiguration
- GetFunctionConfiguration
- CreateFunction

Contoh AWS CLI perintah berikut memungkinkan penelusuran aktif pada fungsi bernama my-function.

```
aws lambda update-function-configuration --function-name my-function \
--tracing-config Mode=Active
```

Mode penelusuran adalah bagian dari konfigurasi khusus versi saat Anda memublikasikan versi fungsi Anda. Anda tidak dapat mengubah mode pelacakan pada versi yang telah diterbitkan.

Mengaktifkan penelusuran dengan CloudFormation

Untuk mengaktifkan penelusuran pada `AWS::Lambda::Function` sumber daya dalam CloudFormation templat, gunakan `TracingConfig` properti.

Example fungsi-inline.yml. — Konfigurasi penelusuran

```
Resources:
  function:
    Type: AWS::Lambda::Function
    Properties:
      TracingConfig:
        Mode: Active
      ...
```

Untuk sumber `AWS::Serverless::Function` daya AWS Serverless Application Model (AWS SAM), gunakan `Tracing` properti.

Example template.yml — Konfigurasi penelusuran

```
Resources:
  function:
    Type: AWS::Serverless::Function
```

```
    Properties:
      Tracing: Active
      ...
```

Menafsirkan jejak X-Ray

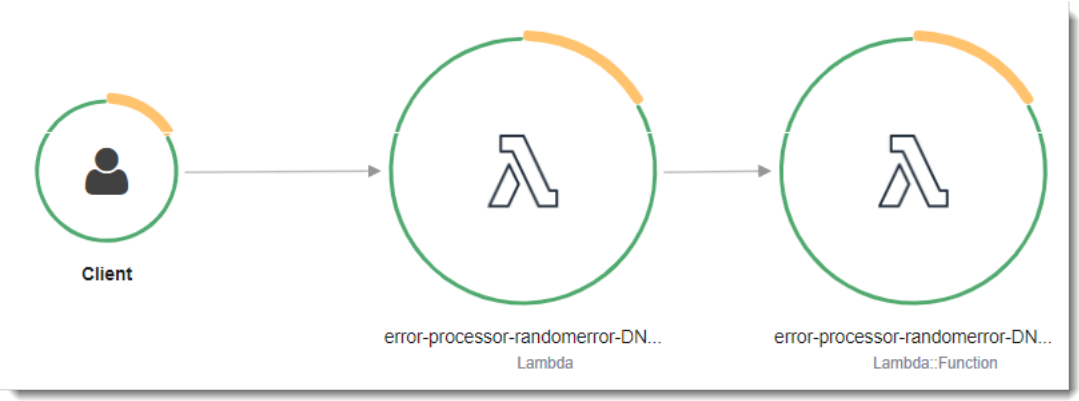
Fungsi Acara Anda memerlukan izin untuk mengunggah data jejak ke X-Ray. Saat Anda mengaktifkan penelusuran di konsol Lambda, Lambda menambahkan izin yang diperlukan ke peran eksekusi Fungsi Acara Anda. Atau, tambahkan kebijakan AWSXRayDaemonWriteAccess ke peran eksekusi.

Setelah mengonfigurasi penelusuran aktif, Anda dapat mengamati permintaan tertentu melalui aplikasi Anda. Grafik X-Ray layanan menunjukkan informasi tentang aplikasi Anda dan semua komponennya. Contoh berikut menunjukkan aplikasi dengan dua fungsi. Fungsi utama memproses kejadian dan terkadang mengembalikan kesalahan. Fungsi kedua di bagian atas memproses kesalahan yang muncul di grup log pertama dan menggunakan AWS SDK untuk memanggil X-Ray, Amazon Simple Storage Service (Amazon S3) Service, dan Amazon Logs. CloudWatch

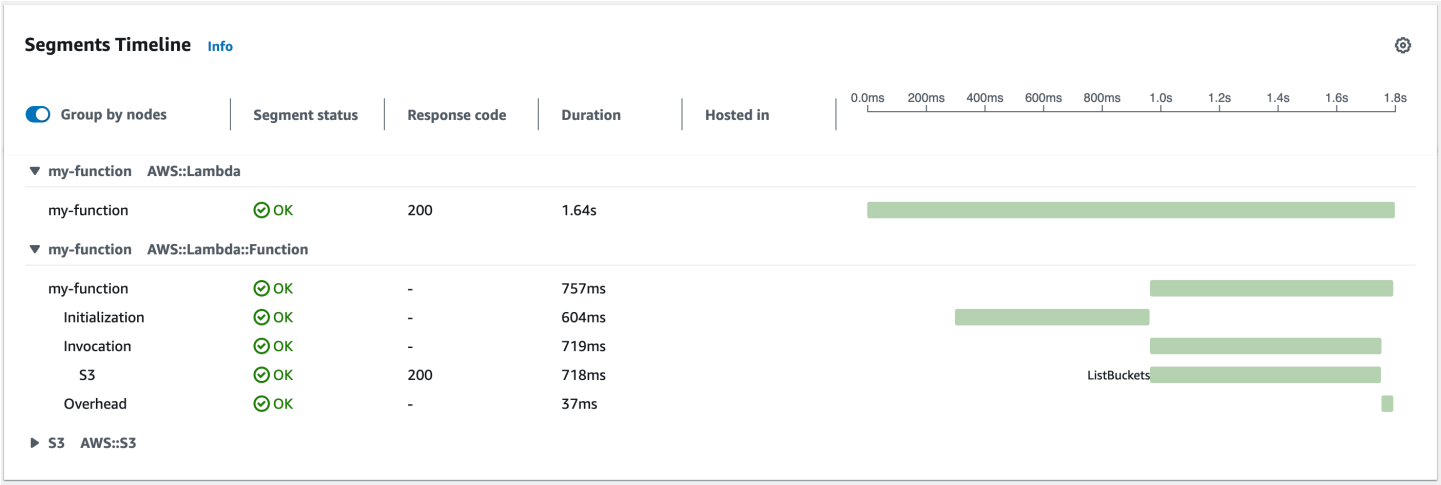


X-Ray tidak melacak semua permintaan ke aplikasi Anda. X-Ray menerapkan algoritma pengambilan sampel untuk memastikan bahwa penelusuran efisien, sambil tetap memberikan sampel yang representatif dari semua permintaan. Tingkat pengambilan sampel adalah 1 permintaan per detik dan 5 persen dari permintaan tambahan. Anda tidak dapat mengonfigurasi laju X-Ray pengambilan sampel untuk fungsi Anda.

Di X-Ray, jejak mencatat informasi tentang permintaan yang diproses oleh satu atau beberapa layanan. Lambda mencatat 2 segmen per jejak, yang menciptakan dua node pada grafik layanan. Gambar berikut menyoroti dua node ini:



Node pertama di sebelah kiri mewakili layanan Lambda, yang menerima permintaan pemanggilan. Node kedua mewakili fungsi Lambda spesifik Anda. Contoh berikut menunjukkan jejak dengan dua segmen ini. Keduanya bernama fungsi saya, tetapi yang satu memiliki asal `AWS::Lambda` dan yang lainnya memiliki asal usul. `AWS::Lambda::Function` Jika `AWS::Lambda` segmen menunjukkan kesalahan, layanan Lambda mengalami masalah. Jika `AWS::Lambda::Function` segmen menunjukkan kesalahan, fungsi Anda mengalami masalah.



Contoh ini memperluas `AWS::Lambda::Function` segmen untuk menunjukkan tiga subsegmennya.

> **Note**
>
> AWS saat ini menerapkan perubahan pada layanan Lambda. Karena perubahan ini, Anda mungkin melihat perbedaan kecil antara struktur dan konten pesan log sistem dan segmen pelacakan yang dipancarkan oleh Fungsi Peristiwa Lambda yang berbeda di Anda. Akun AWS

Contoh jejak yang ditunjukkan di sini menggambarkan segmen fungsi gaya lama. Perbedaan antara segmen gaya lama dan baru dijelaskan dalam paragraf berikut.
Perubahan ini akan diterapkan selama beberapa minggu mendatang, dan semua fungsi di semua Wilayah AWS kecuali China GovCloud dan wilayah akan bertransisi untuk menggunakan pesan log format baru dan segmen pelacakan.

Segmen fungsi gaya lama berisi subsegmen berikut:

- Inisialisasi – Mewakili waktu yang dihabiskan untuk memuat fungsi dan menjalankan kode inisialisasi. Subsegmen ini hanya muncul untuk peristiwa pertama yang diproses oleh setiap instance fungsi Anda.
- Doa - Merupakan waktu yang dihabiskan untuk menjalankan kode handler Anda.
- Overhead - Merupakan waktu yang dihabiskan runtime Lambda untuk mempersiapkan diri untuk menangani acara berikutnya.

Segmen fungsi gaya baru tidak berisi subsegmen. `Invocation` Sebagai gantinya, subsegmen pelanggan dilampirkan langsung ke segmen fungsi. Untuk informasi lebih lanjut tentang struktur segmen fungsi gaya lama dan baru, lihat. the section called “Memahami X-Ray jejak”

Anda juga dapat melakukan instrumentasi klien HTTP, merekam kueri SQL, dan membuat subsegmen khusus dengan anotasi dan metadata. Untuk informasi selengkapnya, lihat AWS X-Ray SDK for .NETdi Panduan AWS X-Ray Pengembang.

Harga

Anda dapat menggunakan X-Ray tracing gratis setiap bulan hingga batas tertentu sebagai bagian dari Tingkat AWS Gratis. Di luar ambang batas itu, X-Ray biaya untuk penyimpanan dan pengambilan jejak. Untuk informasi lebih lanjut, lihat Harga AWS X-Ray.

AWS Lambda Pengujian Fungsi Acara di C

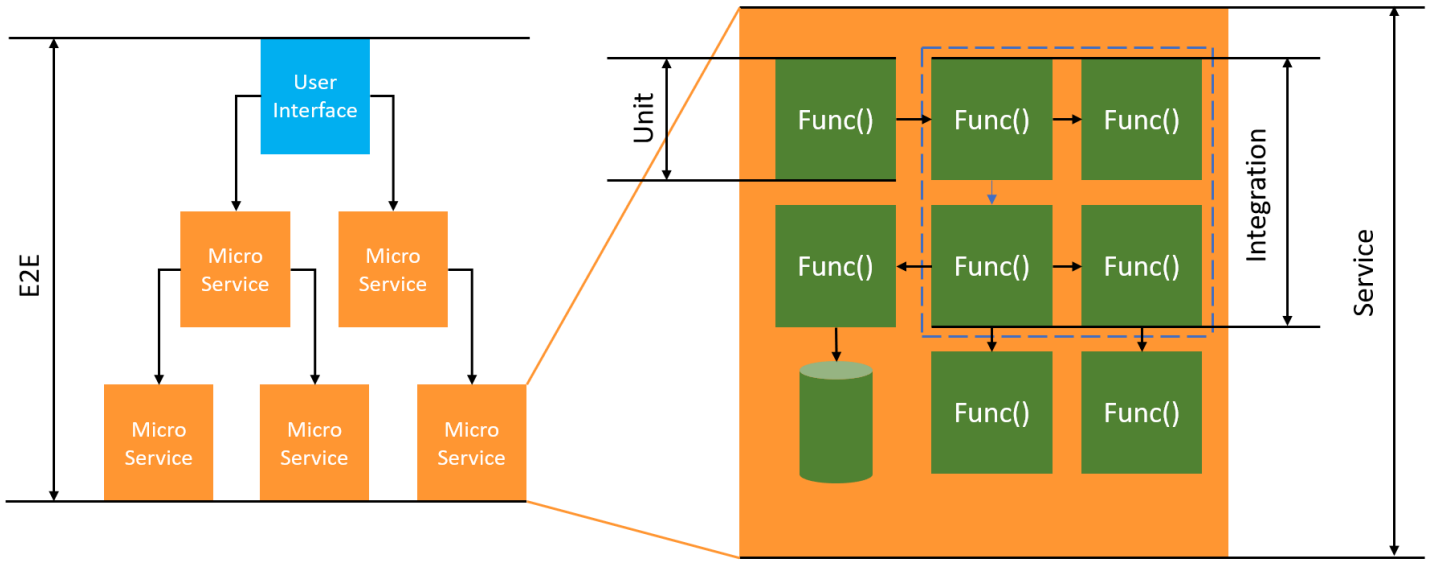
> **Note**
>
> Lihat bagian Fungsi pengujian untuk pengenalan lengkap tentang teknik dan praktik terbaik untuk menguji solusi tanpa server.

Menguji fungsi tanpa server menggunakan jenis dan teknik pengujian tradisional, tetapi Anda juga harus mempertimbangkan pengujian aplikasi tanpa server secara keseluruhan. Cloud-based tes memberikan ukuran kualitas yang paling akurat dari fungsi dan aplikasi tanpa server Anda.

Arsitektur aplikasi tanpa server mencakup layanan terkelola yang menyediakan fungsionalitas aplikasi penting melalui panggilan API. Untuk alasan ini, siklus pengembangan Anda harus menyertakan pengujian otomatis yang memverifikasi fungsionalitas saat fungsi dan layanan Anda berinteraksi.

Jika Anda tidak membuat pengujian berbasis cloud, Anda dapat mengalami masalah karena perbedaan antara lingkungan lokal dan lingkungan yang diterapkan. Proses integrasi berkelanjutan Anda harus menjalankan pengujian terhadap serangkaian sumber daya yang disediakan di cloud sebelum mempromosikan kode Anda ke lingkungan penerapan berikutnya, seperti QA, Staging, atau Production.

Lanjutkan membaca panduan singkat ini untuk mempelajari strategi pengujian untuk aplikasi tanpa server, atau kunjungi repositori Sampel Uji Tanpa Server untuk menyelami contoh-contoh praktis, khusus untuk bahasa dan runtime pilihan Anda.



Untuk pengujian tanpa server, Anda masih menulis pengujian unit, integrasi, dan ujung-ke-ujung.

- Tes unit - Tes yang dijalankan terhadap blok kode yang terisolasi. Misalnya, memverifikasi logika bisnis untuk menghitung biaya pengiriman yang diberikan item dan tujuan tertentu.
- Tes integrasi - Tes yang melibatkan dua atau lebih komponen atau layanan yang berinteraksi, biasanya di lingkungan cloud. Misalnya, memverifikasi fungsi memproses peristiwa dari antrian.
- End-to-end tes - Tes yang memverifikasi perilaku di seluruh aplikasi. Misalnya, memastikan infrastruktur diatur dengan benar dan bahwa peristiwa mengalir antar layanan seperti yang diharapkan untuk merekam pesanan pelanggan.

Menguji aplikasi tanpa server Anda

Anda biasanya menggunakan campuran pendekatan untuk menguji kode aplikasi tanpa server Anda, termasuk pengujian di cloud, pengujian dengan tiruan, dan kadang-kadang menguji dengan emulator.

Pengujian di cloud

Pengujian di cloud sangat berharga untuk semua fase pengujian, termasuk pengujian unit, pengujian integrasi, dan pengujian ujung ke ujung. Anda menjalankan pengujian terhadap kode yang diterapkan di cloud dan berinteraksi dengan layanan berbasis cloud. Pendekatan ini memberikan ukuran kualitas kode Anda yang paling akurat.

Cara mudah untuk men-debug Fungsi Acara Lambda Anda di cloud adalah melalui konsol dengan acara pengujian. Peristiwa pengujian adalah input JSON ke fungsi Anda. Jika fungsi Anda tidak memerlukan input, acara dapat berupa dokumen (`{}`) JSON kosong. Konsol menyediakan

contoh peristiwa untuk berbagai integrasi layanan. Setelah membuat acara di konsol, Anda dapat membagikannya dengan tim Anda untuk membuat pengujian lebih mudah dan konsisten.

> **Note**
>
> Menguji fungsi di konsol adalah cara cepat untuk memulai, tetapi mengotomatiskan siklus pengujian Anda memastikan kualitas aplikasi dan kecepatan pengembangan.

Alat pengujian

Untuk mempercepat siklus pengembangan Anda, ada sejumlah alat dan teknik yang dapat Anda gunakan saat menguji fungsi Anda. Misalnya, mode AWS SAM Akselerasi dan AWS CDK tonton mengurangi waktu yang diperlukan untuk memperbarui lingkungan cloud.

Cara Anda mendefinisikan kode fungsi Lambda Anda membuatnya mudah untuk menambahkan pengujian unit. Lambda membutuhkan konstruktor publik tanpa parameter untuk menginisialisasi kelas Anda. Memperkenalkan konstruktor internal kedua memberi Anda kendali atas dependensi yang digunakan aplikasi Anda.

```
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer

namespace GetProductHandler;

public class Function
{
    private readonly IDatabaseRepository _repo;

    public Function(): this(null)
    {
    }

    internal Function(IDatabaseRepository repo)
    {
        this._repo = repo ?? new DatabaseRepository();
    }

    public async Task<APIGatewayProxyResponse> FunctionHandler(APIGatewayProxyRequest
 request)
    {
```

```
        var id = request.PathParameters["id"];

        var databaseRecord = await this._repo.GetById(id);

        return new APIGatewayProxyResponse
        {
            StatusCode = (int)HttpStatusCode.OK,
            Body = JsonSerializer.Serialize(databaseRecord)
        };
    }
}
```

Untuk menulis tes untuk fungsi ini, Anda dapat menginisialisasi instance baru `Function` kelas Anda dan meneruskan implementasi tiruan dari. `IDatabaseRepository` Contoh di bawah ini menggunakan `XUnit`oq,, dan `FluentAssertions` untuk menulis tes sederhana memastikan `FunctionHandler` pengembalian kode status 200.

```
using Xunit;
using Moq;
using FluentAssertions;

public class FunctionTests
{
    [Fact]
    public async Task TestLambdaHandler_WhenInputIsValid_ShouldReturn200StatusCode()
    {
        // Arrange
        var mockDatabaseRepository = new Mock<IDatabaseRepository>();

        var functionUnderTest = new Function(mockDatabaseRepository.Object);

        // Act
        var response = await functionUnderTest.FunctionHandler(new
 APIGatewayProxyRequest());

        // Assert
        response.StatusCode.Should().Be(200);
    }
}
```

Untuk contoh yang lebih rinci, termasuk contoh pengujian asinkron, lihat repositori sampel pengujian .NET pada. GitHub

Membangun Fungsi Acara Lambda dengan PowerShell

Bagian berikut menjelaskan bagaimana pola pemrograman umum dan konsep inti berlaku ketika Anda membuat kode Fungsi Peristiwa Lambda. PowerShell

Untuk informasi tentang objek PowerShell konteks, lihatthe section called “Konteks”.

Lambda menyediakan contoh aplikasi berikut untuk: PowerShell

- blank-powershell — PowerShell Fungsi yang menunjukkan penggunaan logging, variabel lingkungan, dan SDK. AWS

Sebelum memulai, Anda harus terlebih dahulu mengatur lingkungan PowerShell pengembangan. Untuk petunjuk tentang cara melakukannya, lihat Menyiapkan Lingkungan PowerShell Pembangunan.

Untuk mempelajari cara menggunakan AWSLambdaPSCore modul untuk mengunduh PowerShell proyek sampel dari templat, membuat paket PowerShell penerapan, dan menerapkan PowerShell fungsi ke AWS Cloud, lihat. Menyebarkan Fungsi Acara PowerShell Lambda dengan arsip file.zip

Lambda menyediakan runtime berikut untuk bahasa.NET:

Nama	Pengidentifikasi	Sistem operasi	Tanggal pengusangan	Buat fungsi blok	Pembaruan fungsi blok
.NET 10	`dotnet10`	Amazon Linux 2023	Nov 14, 2028	14 Des 2028	15 Jan 2029
.NET 9 (hanya wadah)	`dotnet9`	Amazon Linux 2023	Nov 10, 2026	Tidak dijadwalkan	Tidak dijadwalkan
.NET 8	`dotnet8`	Amazon Linux 2023	Nov 10, 2026	Jul 29, 2027	Agustus 31, 2027

Topik

- Menyiapkan Lingkungan PowerShell Pembangunan
- Menyebarkan Fungsi Acara PowerShell Lambda dengan arsip file.zip

- Tentukan penangan Fungsi Peristiwa Lambda di PowerShell
- Menggunakan objek konteks Lambda untuk mengambil informasi Fungsi Acara PowerShell
- Log dan pantau Fungsi Acara Powershell Lambda

Menyiapkan Lingkungan PowerShell Pembangunan

Lambda menyediakan seperangkat alat dan pustaka untuk runtime. PowerShell Untuk petunjuk penginstalan, lihat Alat Lambda untuk PowerShell aktif. GitHub

AWSLambdaPSCore Modul ini mencakup cmdlet berikut untuk membantu penulis dan menerbitkan Fungsi Acara PowerShell Lambda:

- Get-AWSPowerShellLambdaTemplate— Mengembalikan daftar template memulai.
- New-AWSPowerShellLambda— Membuat PowerShell skrip awal berdasarkan template.
- Publish-AWSPowerShellLambda— Menerbitkan PowerShell skrip yang diberikan ke Lambda.
- New-AWSPowerShellLambdaPackage— Membuat paket penyebaran Lambda yang dapat Anda gunakan dalam CI/CD sistem untuk penerapan.

Menyebarkan Fungsi Acara PowerShell Lambda dengan arsip file.zip

Paket penerapan untuk PowerShell runtime berisi PowerShell skrip Anda, PowerShell modul yang diperlukan untuk PowerShell skrip Anda, dan rakitan yang diperlukan untuk meng-host Core. PowerShell

Membuat Fungsi Acara Lambda

Untuk mulai menulis dan menjalankan PowerShell skrip dengan Lambda, Anda dapat menggunakan `New-AWSPowerShellLambda` cmdlet untuk membuat skrip pemula berdasarkan templat. Anda dapat menggunakan cmdlet `Publish-AWSPowerShellLambda` untuk men-deploy skrip Anda ke Lambda. Kemudian, Anda dapat menguji skrip Anda melalui baris perintah atau konsol Lambda.

Untuk membuat PowerShell skrip baru, unggah, dan uji, lakukan hal berikut:

1. Untuk menampilkan daftar templat yang tersedia, jalankan perintah berikut:

```
PS C:\> Get-AWSPowerShellLambdaTemplate

Template               Description
--------               -----------
Basic                  Bare bones script
CodeCommitTrigger      Script to process AWS CodeCommit Triggers
...
```

2. Untuk membuat skrip sampel berdasarkan templat `Basic`, jalankan perintah berikut:

```
New-AWSPowerShellLambda -ScriptName MyFirstPSScript -Template Basic
```

   File baru bernama `MyFirstPSScript.ps1` dibuat di subdirektori baru dari direktori saat ini. Nama direktori didasarkan pada parameter `-ScriptName`. Anda dapat menggunakan parameter `-Directory` untuk memilih direktori alternatif.

   Anda dapat melihat file baru tersebut memiliki konten berikut:

```
# PowerShell script file to run as a Lambda function
#
# When executing in Lambda the following variables are predefined.
#   $LambdaInput - A PSObject that contains the Lambda function input data.
#   $LambdaContext - An Amazon.Lambda.Core.ILambdaContext object that contains
 information about the currently running Lambda environment.
```

```
#
# The last item in the PowerShell pipeline is returned as the result of the Lambda
 function.
#
# To include PowerShell modules with your Lambda function, like the
 AWSPowerShell.NetCore module, add a "#Requires" statement
# indicating the module and version.

#Requires -Modules @{ModuleName='AWSPowerShell.NetCore';ModuleVersion='3.3.618.0'}

# Uncomment to send the input to CloudWatch Logs
# Write-Host (ConvertTo-Json -InputObject $LambdaInput -Compress -Depth 5)
```

3. Untuk melihat bagaimana pesan log dari PowerShell skrip Anda dikirim ke Amazon CloudWatch Logs, batalkan komentar pada `Write-Host` baris skrip sampel.

   Untuk mendemonstrasikan bagaimana Anda dapat mengembalikan data dari fungsi Lambda Anda, tambahkan garis baru di akhir skrip dengan `$PSVersionTable`. Ini menambah `$PSVersionTable` ke PowerShell pipa. Setelah PowerShell skrip selesai, objek terakhir dalam PowerShell pipeline adalah data pengembalian untuk fungsi Lambda. `$PSVersionTable`adalah variabel PowerShell global yang juga memberikan informasi tentang lingkungan berjalan.

   Setelah membuat perubahan ini, dua baris terakhir dari skrip sampel akan terlihat seperti ini:

   ```
   Write-Host (ConvertTo-Json -InputObject $LambdaInput -Compress -Depth 5)
   $PSVersionTable
   ```

4. Setelah mengedit file `MyFirstPSScript.ps1`, ubah direktori ke lokasi skrip. Kemudian jalankan perintah berikut untuk menerbitkan skrip ke Lambda:

   ```
   Publish-AWSPowerShellLambda -ScriptPath .\MyFirstPSScript.ps1 -Name
    MyFirstPSScript -Region us-east-2
   ```

   Perhatikan bahwa parameter `-Name` menentukan nama fungsi Lambda, yang muncul di konsol Lambda. Anda dapat menggunakan fungsi ini untuk mengaktifkan skrip Anda secara manual.

5. Memanggil fungsi Anda menggunakan `invoke` perintah AWS Command Line Interface (AWS CLI).

   ```
   > aws lambda invoke --function-name MyFirstPSScript out
   ```

Tentukan penangan Fungsi Peristiwa Lambda di PowerShell

Ketika fungsi Lambda dipanggil, penangan Lambda memanggil skrip. PowerShell

Saat PowerShell skrip dipanggil, variabel berikut telah ditentukan sebelumnya:

- *$LambdaInput*— PSObject yang berisi input ke handler. Input ini dapat berupa data peristiwa (diterbitkan oleh sumber peristiwa) atau input kustom yang Anda berikan, seperti string atau objek data kustom apa pun.
- *$LambdaContext* Amazon.Lambda.Core.ILambdaContext Objek yang dapat Anda gunakan untuk mengakses informasi tentang pemanggilan saat ini—seperti nama fungsi saat ini, batas memori, waktu eksekusi yang tersisa, dan pencatatan.

Misalnya, perhatikan PowerShell contoh kode berikut.

```
#Requires -Modules @{ModuleName='AWSPowerShell.NetCore';ModuleVersion='3.3.618.0'}
Write-Host 'Function Name:' $LambdaContext.FunctionName
```

Script ini mengembalikan FunctionName properti yang diperoleh dari LambdaContext variabel $.

> **Note**
>
> Anda harus menggunakan `#Requires` pernyataan dalam PowerShell skrip Anda untuk menunjukkan modul yang menjadi sandaran skrip Anda. Pernyataan ini melakukan dua tugas penting. 1) Ini berkomunikasi dengan pengembang lain modul mana yang digunakan skrip, dan 2) mengidentifikasi modul dependen yang perlu dikemas AWS PowerShell alat dengan skrip, sebagai bagian dari penerapan. Untuk informasi selengkapnya tentang `#Requires` pernyataan di PowerShell, lihat Tentang membutuhkan. Untuk informasi selengkapnya tentang paket PowerShell penerapan, lihatMenyebarkan Fungsi Acara PowerShell Lambda dengan arsip file.zip.
>
> Saat fungsi PowerShell Lambda Anda menggunakan AWS PowerShell cmdlet, pastikan untuk menetapkan `#Requires` pernyataan yang mereferensikan `AWSPowerShell.NetCore` modul, yang mendukung PowerShell Core—dan bukan modul, yang hanya mendukung Windows. `AWSPowerShell` PowerShell Selain itu, pastikan untuk menggunakan versi 3.3.270.0 atau yang lebih baru dari `AWSPowerShell.NetCore` yang mengoptimalkan proses impor cmdlet. Jika Anda menggunakan versi lama, Anda

akan mengalami proses mulai dari awal dalam waktu yang lebih lama. Untuk informasi selengkapnya, lihat AWS Alat untuk PowerShell.

Mengembalikan data

Beberapa invokasi Lambda dimaksudkan untuk mengembalikan data ke pemanggilnya. Misalnya, jika invokasi adalah sebagai respons terhadap permintaan web yang berasal dari API Gateway, fungsi Lambda kita harus mengembalikan respons tersebut. Untuk PowerShell Lambda, objek terakhir yang ditambahkan ke PowerShell pipeline adalah data pengembalian dari pemanggilan Lambda. Jika objek berupa string, data akan dikembalikan apa adanya. Jika tidak, objek dikonversi ke JSON dengan menggunakan cmdlet `ConvertTo-Json`.

Misalnya, perhatikan PowerShell pernyataan berikut, yang `$PSVersionTable` menambah PowerShell pipa:

```
$PSVersionTable
```

Setelah PowerShell skrip selesai, objek terakhir dalam PowerShell pipeline adalah data pengembalian untuk fungsi Lambda. `$PSVersionTable`adalah variabel PowerShell global yang juga memberikan informasi tentang lingkungan berjalan.

Menggunakan objek konteks Lambda untuk mengambil informasi Fungsi Acara PowerShell

Ketika Lambda menjalankan fungsi Anda, hal ini melewati informasi konteks dengan membuat variabel `$LambdaContext` tersedia untuk handler. Variabel ini menyediakan metode dan properti dengan informasi tentang lingkungan invokasi, fungsi, dan eksekusi.

Properti konteks

- `FunctionName` – Nama fungsi Lambda.
- `FunctionVersion` – Versi fungsi.
- `InvokedFunctionArn` – Amazon Resource Name (ARN) yang digunakan untuk memicu fungsi. Menunjukkan jika pemicu menyebutkan nomor versi atau alias.
- `MemoryLimitInMB` – Jumlah memori yang dialokasikan untuk fungsi tersebut.
- `AwsRequestId` – Pengidentifikasi permintaan invokasi.
- `LogGroupName` – Grup log untuk fungsi.
- `LogStreamName` – Aliran log untuk instans fungsi.
- `RemainingTime` – Jumlah milidetik yang tersisa sebelum waktu eksekusi habis.
- `Identity` – (aplikasi seluler) Informasi tentang identitas Amazon Cognito yang mengesahkan permintaan.
- `ClientContext` – (aplikasi seluler) Konteks klien yang disediakan untuk Lambda oleh aplikasi klien.
- `Logger` – Objek logger untuk fungsi.

Cuplikan PowerShell kode berikut menunjukkan fungsi handler sederhana yang mencetak beberapa informasi konteks.

```
#Requires -Modules @{ModuleName='AWSPowerShell.NetCore';ModuleVersion='3.3.618.0'}
Write-Host 'Function name:' $LambdaContext.FunctionName
Write-Host 'Remaining milliseconds:' $LambdaContext.RemainingTime.TotalMilliseconds
Write-Host 'Log group name:' $LambdaContext.LogGroupName
Write-Host 'Log stream name:' $LambdaContext.LogStreamName
```

Log dan pantau Fungsi Acara Powershell Lambda

AWS Lambda secara otomatis memonitor Fungsi Acara Lambda atas nama Anda dan mengirim log ke Amazon. CloudWatch Fungsi Anda dilengkapi dengan grup CloudWatch log Log dan aliran log untuk setiap instance fungsi Anda. Lingkungan runtime Lambda mengirimkan detail tentang setiap invokasi ke pengaliran log, dan menyampaikan log serta output lain dari kode fungsi Anda. Untuk informasi selengkapnya, lihat Mengirim log Fungsi Peristiwa Lambda ke Log CloudWatch.

Halaman ini menjelaskan cara menghasilkan keluaran log dari kode fungsi Lambda Anda, dan mengakses log menggunakan AWS Command Line Interface, konsol Lambda, atau konsol. CloudWatch

Bagian-bagian

- Membuat fungsi yang mengembalikan log
- Melihat log di konsol Lambda
- Melihat log di CloudWatch konsol
- Melihat log menggunakan AWS Command Line Interface (AWS CLI)
- Menghapus log

Membuat fungsi yang mengembalikan log

Untuk mengeluarkan log dari kode fungsi Anda, Anda dapat menggunakan cmdlet pada Microsoft.PowerShell.Utility, atau modul logging apa pun yang menulis ke `stdout` atau. `stderr` Contoh berikut menggunakan `Write-Host`.

Example function/Handler.ps1 — Penebangan

```
#Requires -Modules @{ModuleName='AWSPowerShell.NetCore';ModuleVersion='3.3.618.0'}
Write-Host `## Environment variables
Write-Host AWS_LAMBDA_FUNCTION_VERSION=$Env:AWS_LAMBDA_FUNCTION_VERSION
Write-Host AWS_LAMBDA_LOG_GROUP_NAME=$Env:AWS_LAMBDA_LOG_GROUP_NAME
Write-Host AWS_LAMBDA_LOG_STREAM_NAME=$Env:AWS_LAMBDA_LOG_STREAM_NAME
Write-Host AWS_EXECUTION_ENV=$Env:AWS_EXECUTION_ENV
Write-Host AWS_LAMBDA_FUNCTION_NAME=$Env:AWS_LAMBDA_FUNCTION_NAME
Write-Host PATH=$Env:PATH
Write-Host `## Event
Write-Host (ConvertTo-Json -InputObject $LambdaInput -Compress -Depth 3)
```

Example format log

```
START RequestId: 56639408-xmpl-435f-9041-ac47ae25ceed Version: $LATEST
Importing module ./Modules/AWSPowerShell.NetCore/3.3.618.0/AWSPowerShell.NetCore.psd1
[Information] - ## Environment variables
[Information] - AWS_LAMBDA_FUNCTION_VERSION=$LATEST
[Information] - AWS_LAMBDA_LOG_GROUP_NAME=/aws/lambda/blank-powershell-
function-18CIXMPLHFAJJ
[Information] - AWS_LAMBDA_LOG_STREAM_NAME=2020/04/01/
[$LATEST]53c5xmpl52d64ed3a744724d9c201089
[Information] - AWS_EXECUTION_ENV=AWS_Lambda_dotnet6_powershell_1.0.0
[Information] - AWS_LAMBDA_FUNCTION_NAME=blank-powershell-function-18CIXMPLHFAJJ
[Information] - PATH=/var/lang/bin:/usr/local/bin:/usr/bin/:/bin:/opt/bin
[Information] - ## Event
[Information] -
{
    "Records": [
        {
            "messageId": "19dd0b57-b21e-4ac1-bd88-01bbb068cb78",
            "receiptHandle": "MessageReceiptHandle",
            "body": "Hello from SQS!",
            "attributes": {
                "ApproximateReceiveCount": "1",
                "SentTimestamp": "1523232000000",
                "SenderId": "123456789012",
                "ApproximateFirstReceiveTimestamp": "1523232000001"
            },
            ...
END RequestId: 56639408-xmpl-435f-9041-ac47ae25ceed
REPORT RequestId: 56639408-xmpl-435f-9041-ac47ae25ceed Duration: 3906.38 ms Billed
 Duration: 9867 ms Memory Size: 512 MB Max Memory Used: 367 MB Init Duration: 5960.19
 ms
XRAY TraceId: 1-5e843da6-733cxmple7d0c3c020510040 SegmentId: 3913xmpl20999446 Sampled:
 true
```

Runtime .NET mencatat baris `START`, `END`, dan `REPORT` untuk setiap invokasi. Baris laporan memberikan perincian berikut.

Laporkan bidang data baris

- RequestId – ID permintaan unik untuk invokasi.

- Durasi – Jumlah waktu yang digunakan oleh metode handler fungsi Anda gunakan untuk memproses peristiwa.
- Durasi yang Ditagih – Jumlah waktu yang ditagihkan untuk invokasi.
- Ukuran Memori – Jumlah memori yang dialokasikan untuk fungsi.
- Memori Maks yang Digunakan – Jumlah memori yang digunakan oleh fungsi. Saat pemanggilan berbagi lingkungan eksekusi, Lambda melaporkan memori maksimum yang digunakan di semua pemanggilan. Perilaku ini dapat menghasilkan nilai yang dilaporkan lebih tinggi dari yang diharapkan.
- Durasi Init – Untuk permintaan pertama yang dilayani, lama waktu yang diperlukan runtime untuk memuat fungsi dan menjalankan kode di luar metode handler.
- XRAY TraceId — Untuk permintaan yang dilacak, ID AWS X-Ray jejak.
- SegmentId— Untuk permintaan yang dilacak, ID X-Ray segmen.
- Diambil Sampel – Untuk permintaan yang dilacak, hasil pengambilan sampel.

Melihat log di konsol Lambda

Anda dapat menggunakan konsol Lambda untuk melihat keluaran log setelah Anda menjalankan Fungsi Peristiwa Lambda.

Jika kode Anda dapat diuji dari editor Kode tertanam, Anda menemukan log dalam hasil eksekusi. Saat Anda menggunakan fitur pengujian konsol untuk menjalankan fungsi, Anda menemukan Keluaran Log di bagian Detail.

Melihat log di CloudWatch konsol

Anda dapat menggunakan CloudWatch konsol Amazon untuk melihat log untuk semua pemanggilan Fungsi Peristiwa Lambda.

Untuk melihat log di CloudWatch konsol

1. Buka halaman Grup log di CloudWatch konsol.
2. Pilih grup log untuk fungsi Anda (/aws/lambda/***your-function-name***).
3. Pilih pengaliran log.

Setiap aliran log sesuai dengan instans fungsi Anda. Aliran log muncul saat Anda memperbarui fungsi Lambda, dan saat instance tambahan dibuat untuk menangani pemanggilan bersamaan.

Untuk menemukan log untuk pemanggilan tertentu, kami sarankan untuk menginstrumentasi fungsi Anda dengan. AWS X-Ray X-Ray mencatat detail tentang permintaan dan aliran log dalam jejak.

Melihat log menggunakan AWS Command Line Interface (AWS CLI)

AWS CLI Ini adalah alat sumber terbuka yang dapat Anda gunakan untuk berinteraksi dengan AWS layanan menggunakan perintah di shell baris perintah Anda. Untuk menyelesaikan langkah-langkah di bagian ini, Anda harus memiliki AWS CLI versi 2.

Anda dapat menggunakan AWS CLI untuk mengambil log untuk invokasi menggunakan opsi perintah `--log-type`. Respons berisi bidang `LogResult` yang memuat hingga 4 KB log berkode base64 dari invokasi.

Example mengambil ID log

Contoh berikut menunjukkan cara mengambil ID log dari `LogResult` untuk fungsi bernama `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

Anda akan melihat output berikut:

```
{
    "StatusCode": 200,
    "LogResult":
 "U1RBUlQgUmVxdWVzdElkOiA4N2QwNDRiOC1mMTU0LTExZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "ExecutedVersion": "$LATEST"
}
```

Example mendekode log

Pada prompt perintah yang sama, gunakan utilitas `base64` untuk mendekodekan log. Contoh berikut menunjukkan cara mengambil log berkode base64 untuk `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail \
--query 'LogResult' --output text --cli-binary-format raw-in-base64-out | base64 --
decode
```

cli-binary-formatOpsi ini diperlukan jika Anda menggunakan AWS CLI versi 2. Untuk menjadikan ini pengaturan default, jalankan`aws configure set cli-binary-format raw-in-base64-out`.

Untuk informasi selengkapnya, lihat opsi baris perintah global yang AWS CLI didukung di Panduan AWS Command Line Interface Pengguna untuk Versi 2.

Anda akan melihat output berikut:

```
START RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Version: $LATEST
"AWS_SESSION_TOKEN": "AgoJb3JpZ2luX2VjELj...", "_X_AMZN_TRACE_ID": "Root=1-5d02e5ca-
f5792818b6fe8368e5b51d50;Parent=191db58857df8395;Sampled=0"",ask/lib:/opt/lib",
END RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8
REPORT RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8  Duration: 79.67 ms      Billed
 Duration: 80 ms         Memory Size: 128 MB     Max Memory Used: 73 MB
```

Utilitas `base64` tersedia di Linux, macOS, dan Ubuntu pada Windows. Pengguna macOS mungkin harus menggunakan `base64 -D`.

Example Skrip get-logs.sh

Pada prompt perintah yang sama, gunakan script berikut untuk mengunduh lima peristiwa log terakhir. Skrip menggunakan `sed` untuk menghapus kutipan dari file output, dan akan tidur selama 15 detik untuk memberikan waktu agar log tersedia. Output mencakup respons dari Lambda dan output dari perintah `get-log-events`.

Salin konten dari contoh kode berikut dan simpan dalam direktori proyek Lambda Anda sebagai `get-logs.sh`.

cli-binary-formatOpsi ini diperlukan jika Anda menggunakan AWS CLI versi 2. Untuk menjadikan ini pengaturan default, jalankan`aws configure set cli-binary-format raw-in-base64-out`. Untuk informasi selengkapnya, lihat opsi baris perintah global yang AWS CLI didukung di Panduan AWS Command Line Interface Pengguna untuk Versi 2.

```
#!/bin/bash
aws lambda invoke --function-name my-function --cli-binary-format raw-in-base64-out --
payload '{"key": "value"}' out
sed -i'' -e 's/"//g' out
sleep 15
aws logs get-log-events --log-group-name /aws/lambda/my-function --log-stream-
name stream1 --limit 5
```

Example macOS dan Linux (khusus)

Dalam prompt perintah yang sama, pengguna macOS dan Linux mungkin perlu menjalankan perintah berikut untuk memastikan skrip dapat dieksekusi.

```
chmod -R 755 get-logs.sh
```

Example mengambil lima log acara terakhir

Pada prompt perintah yang sama, gunakan skrip berikut untuk mendapatkan lima log acara terakhir.

```
./get-logs.sh
```

Anda akan melihat output berikut:

```
{
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}
{
    "events": [
        {
            "timestamp": 1559763003171,
            "message": "START RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf Version:
 $LATEST\n",
            "ingestionTime": 1559763003309
        },
        {
            "timestamp": 1559763003173,
            "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tENVIRONMENT VARIABLES\r{\r  \"AWS_LAMBDA_FUNCTION_VERSION\": \"$LATEST\",
\r ...",
            "ingestionTime": 1559763018353
        },
        {
            "timestamp": 1559763003173,
            "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tEVENT\r{\r  \"key\": \"value\"\r}\n",
            "ingestionTime": 1559763018353
        },
        {
            "timestamp": 1559763003218,
            "message": "END RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf\n",
            "ingestionTime": 1559763018353
        },
        {
            "timestamp": 1559763003218,
```

```
            "message": "REPORT RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tDuration: 26.73 ms\tBilled Duration: 27 ms \tMemory Size: 128 MB\tMax Memory Used: 75
 MB\t\n",
            "ingestionTime": 1559763018353
        }
    ],
    "nextForwardToken": "f/34783877304859518393868359594929986069206639495374241795",
    "nextBackwardToken": "b/34783877303811383369537420289090800615709599058929582080"
}
```

Menghapus log

Grup log tidak terhapus secara otomatis ketika Anda menghapus suatu fungsi. Untuk menghindari penyimpanan log secara tidak terbatas, hapus kelompok log, atau lakukan konfigurasi periode penyimpanan, yang setelahnya log akan dihapus secara otomatis.

Membangun Fungsi Acara Lambda dengan Rust

Karena Rust mengkompilasi ke kode asli, Anda tidak memerlukan runtime khusus untuk menjalankan kode Rust sebagai Fungsi Peristiwa Lambda. Sebagai gantinya, gunakan klien runtime Rust untuk membangun proyek Anda secara lokal, lalu terapkan ke Lambda menggunakan runtime. OS-only Saat Anda menggunakan OS-only runtime, Lambda secara otomatis menjaga sistem operasi tetap up to date dengan patch terbaru. Untuk informasi tentang bekerja dengan peristiwa HTTP, lihatthe section called “Acara HTTP”. Untuk informasi tentang menggunakan layer dengan Rust, lihatthe section called “Lapisan”.

Alat dan pustaka untuk Rust

- AWS SDK for Rust: AWS SDK for Rust menyediakan Rust API untuk berinteraksi dengan layanan infrastruktur Amazon Web Services.
- Klien runtime Rust untuk Lambda: Klien runtime Rust memudahkan menjalankan fungsi Lambda yang ditulis dalam Rust.
- Cargo Lambda: Ini adalah ekstensi sumber terbuka pihak ketiga untuk alat baris perintah Cargo yang menyederhanakan pembuatan dan penerapan fungsi Rust Lambda.
- Lambda HTTP: Pustaka ini menyediakan pembungkus untuk bekerja dengan peristiwa HTTP.
- Ekstensi Lambda: Pustaka ini menyediakan dukungan untuk menulis Ekstensi Lambda dengan Rust.
- AWS Lambda Peristiwa: Pustaka ini menyediakan definisi tipe untuk integrasi sumber peristiwa umum.

Contoh aplikasi Lambda untuk Rust

- Fungsi Lambda Dasar: Fungsi Rust yang menunjukkan cara memproses peristiwa dasar.
- Fungsi Lambda dengan penanganan kesalahan: Fungsi Rust yang menunjukkan cara menangani kesalahan Rust khusus di Lambda.
- Fungsi Lambda dengan sumber daya bersama: Proyek Rust yang menginisialisasi sumber daya bersama sebelum membuat fungsi Lambda.
- Peristiwa HTTP Lambda: Fungsi Rust yang menangani peristiwa HTTP.
- Peristiwa Lambda HTTP dengan header CORS: Fungsi Rust yang menggunakan Tower untuk menyuntikkan header CORS.
- Lambda REST API: REST API yang menggunakan Axum dan Diesel untuk terhubung ke database PostgreSQL.
- Demo Rust Tanpa Server: Proyek Rust yang menunjukkan penggunaan pustaka Lambda Rust, logging, variabel lingkungan, dan SDK. AWS
- Ekstensi Lambda Dasar: Ekstensi Rust yang menunjukkan cara memproses peristiwa ekstensi dasar.
- Lambda Logs Amazon Data Firehose Extension: Ekstensi Rust yang menunjukkan cara mengirim log Lambda ke Firehose.

Topik

- Tentukan penangan Fungsi Peristiwa Lambda di Rust
- Menggunakan objek konteks Lambda untuk mengambil informasi Rust Event Function
- Memproses peristiwa HTTP dengan Rust
- Terapkan Fungsi Peristiwa Rust Lambda dengan arsip file.zip
- Bekerja dengan lapisan untuk Fungsi Acara Lambda Rust
- Log dan pantau Fungsi Acara Rust Lambda

Tentukan penangan Fungsi Peristiwa Lambda di Rust

Lambda Event Function handler adalah metode dalam kode fungsi Anda yang memproses peristiwa. Saat fungsi Anda diaktifkan, Lambda menjalankan metode handler. Fungsi Anda berjalan sampai handler mengembalikan respons, keluar, atau waktu habis.

Halaman ini menjelaskan cara bekerja dengan penangan Fungsi Peristiwa Lambda di Rust, termasuk inisialisasi proyek, konvensi penamaan, dan praktik terbaik. Halaman ini juga menyertakan contoh fungsi Rust Lambda yang mengambil informasi tentang pesanan, menghasilkan tanda terima file teks, dan menempatkan file ini di bucket Amazon Simple Storage Service (S3). Untuk informasi selengkapnya tentang cara menerapkan fungsi Anda setelah menulisnya, lihatthe section called "Deploy arsip file .zip".

Topik

- Menyiapkan proyek penangan Rust Anda
- Contoh kode Fungsi Peristiwa Lambda Rust
- Definisi kelas yang valid untuk penangan Rust
- Konvensi penamaan handler
- Mendefinisikan dan mengakses objek peristiwa masukan
- Mengakses dan menggunakan objek konteks Lambda
- Menggunakan AWS SDK for Rust di handler Anda
- Mengakses variabel lingkungan
- Menggunakan status bersama
- Praktik terbaik kode untuk Fungsi Acara Lambda Rust

Menyiapkan proyek penangan Rust Anda

Saat bekerja dengan Lambda Event Functions di Rust, prosesnya melibatkan penulisan kode Anda, mengompilasinya, dan menerapkan artefak yang dikompilasi ke Lambda. Cara termudah untuk menyiapkan proyek penangan Lambda di Rust adalah dengan menggunakan AWS Lambda Runtime for Rust. Terlepas dari namanya, AWS Lambda Runtime for Rust bukanlah runtime yang dikelola dalam arti yang sama seperti di Lambda untuk Python, Java, atau. Node.js Sebaliknya, AWS Lambda Runtime for Rust adalah crate (`lambda_runtime`) yang mendukung penulisan fungsi Lambda di Rust dan berinteraksi dengan lingkungan eksekusi. AWS Lambda

Gunakan perintah berikut untuk menginstal Cargo Lambda, ekstensi sumber terbuka pihak ketiga ke alat baris perintah Cargo yang menyederhanakan pembuatan dan penerapan fungsi Rust Lambda:

```
cargo install cargo-lambda
```

Setelah Anda berhasil menginstal`cargo-lambda`, gunakan perintah berikut untuk menginisialisasi proyek penangan fungsi Rust Lambda baru:

```
cargo lambda new example-rust
```

Saat Anda menjalankan perintah ini, antarmuka baris perintah (CLI) menanyakan beberapa pertanyaan tentang fungsi Lambda Anda:

- Fungsi HTTP - Jika Anda bermaksud untuk memanggil fungsi Anda melalui API Gateway atau URL fungsi, jawab Ya. Kalau tidak, jawab No. Dalam kode contoh di halaman ini, kami memanggil fungsi kami dengan acara JSON khusus, jadi kami menjawab No.
- Jenis peristiwa - Jika Anda bermaksud menggunakan bentuk peristiwa yang telah ditentukan untuk menjalankan fungsi Anda, pilih jenis peristiwa yang diharapkan dengan benar. Jika tidak, biarkan opsi ini kosong. Dalam kode contoh di halaman ini, kami memanggil fungsi kami dengan acara JSON khusus, jadi kami membiarkan opsi ini kosong.

Setelah perintah berjalan dengan sukses, masukkan direktori utama proyek Anda:

```
cd example-rust
```

Perintah ini menghasilkan `generic_handler.rs` file dan `main.rs` file di `src` direktori. `generic_handler.rs`Dapat digunakan untuk menyesuaikan event handler generik. `main.rs`File berisi logika aplikasi utama Anda. `Cargo.toml`File berisi metadata tentang paket Anda dan mencantumkan dependensi eksternalnya.

Contoh kode Fungsi Peristiwa Lambda Rust

Contoh berikut kode fungsi Rust Lambda mengambil informasi tentang pesanan, menghasilkan tanda terima file teks, dan menempatkan file ini di bucket Amazon S3.

Example **`main.rs` Fungsi Acara Lambda**

```
use aws_sdk_s3::{Client, primitives::ByteStream};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};
```

```
use serde::{Deserialize, Serialize};
use serde_json::Value;
use std::env;

#[derive(Deserialize, Serialize)]
struct Order {
    order_id: String,
    amount: f64,
    item: String,
}

async fn function_handler(event: LambdaEvent<Value>) -> Result<String, Error> {
    let payload = event.payload;

    // Deserialize the incoming event into Order struct
    let order: Order = serde_json::from_value(payload)?;

    let bucket_name = env::var("RECEIPT_BUCKET")
        .map_err(|_| "RECEIPT_BUCKET environment variable is not set")?;

    let receipt_content = format!(
        "OrderID: {}\nAmount: ${:.2}\nItem: {}",
        order.order_id, order.amount, order.item
    );
    let key = format!("receipts/{}.txt", order.order_id);

    let config =
 aws_config::load_defaults(aws_config::BehaviorVersion::latest()).await;
    let s3_client = Client::new(&config);

    upload_receipt_to_s3(&s3_client, &bucket_name, &key, &receipt_content).await?;

    Ok("Success".to_string())
}

async fn upload_receipt_to_s3(
    client: &Client,
    bucket_name: &str,
    key: &str,
    content: &str,
) -> Result<(), Error> {
    client
        .put_object()
        .bucket(bucket_name)
```

```
        .key(key)
        .body(ByteStream::from(content.as_bytes().to_vec()))  // Fixed conversion
        .content_type("text/plain")
        .send()
        .await?;

    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    run(service_fn(function_handler)).await
}
```

`main.rs`File ini berisi bagian kode berikut:

- `use`pernyataan: Gunakan ini untuk mengimpor peti Rust dan metode yang diperlukan oleh fungsi Lambda Anda.
- `#[derive(Deserialize, Serialize)]`: Tentukan bentuk peristiwa input yang diharapkan dalam struct Rust ini.
- `async fn function_handler(event: LambdaEvent<Value>) -> Result<String, Error>`: Ini adalah metode handler utama, yang berisi logika aplikasi utama Anda.
- `async fn upload_receipt_to_s3 (...)`: Ini adalah metode pembantu yang direferensikan oleh metode utama`function_handler`.
- `#[tokio::main]`: Ini adalah makro yang menandai titik masuk program Rust. Ini juga menyiapkan runtime Tokio, yang memungkinkan `main()` metode Anda menggunakan`async`/`await`dan berjalan secara asinkron.
- `async fn main() -> Result<(), Error>`: `main()` Fungsi ini adalah titik masuk kode Anda. Di dalamnya, kita tentukan `function_handler` sebagai metode handler utama.

Contoh Cargo.toml berkas

`Cargo.toml`File berikut menyertai fungsi ini.

```
[package]
name = "example-rust"
version = "0.1.0"
edition = "2024"
```

```
[dependencies]
aws-config = "1.5.18"
aws-sdk-s3 = "1.78.0"
lambda_runtime = "0.13.0"
serde = { version = "1", features = ["derive"] }
serde_json = "1"
tokio = { version = "1", features = ["full"] }
```

Agar fungsi ini berfungsi dengan baik, peran pelaksanaannya harus memungkinkan `s3:PutObject` tindakan. Juga, pastikan bahwa Anda mendefinisikan variabel `RECEIPT_BUCKET` lingkungan. Setelah pemanggilan berhasil, bucket Amazon S3 harus berisi file tanda terima.

Definisi kelas yang valid untuk penangan Rust

Dalam kebanyakan kasus, tanda tangan penangan Lambda yang Anda tentukan di Rust memiliki format berikut:

```
async fn function_handler(event: LambdaEvent<T>) -> Result<U, Error>
```

Untuk pawang ini:

- Nama handler ini adalah`function_handler`.
- Input tunggal ke handler adalah event, dan bertipe. `LambdaEvent<T>`
  - `LambdaEvent`adalah pembungkus yang berasal dari `lambda_runtime` peti. Menggunakan pembungkus ini memberi Anda akses ke objek konteks, yang mencakup Lambda-specific metadata seperti ID permintaan pemanggilan.
  - Tadalah jenis acara deserialisasi. Misalnya, ini bisa jadi`serde_json::Value`, yang memungkinkan handler untuk mengambil input JSON generik apa pun. Atau, ini bisa berupa tipe seperti `ApiGatewayProxyRequest` jika fungsi Anda mengharapkan tipe input tertentu yang telah ditentukan sebelumnya.
- Jenis pengembalian handler adalah`Result<U, Error>`.
  - Uadalah tipe keluaran deserialisasi. Uharus mengimplementasikan `serde::Serialize` sifat tersebut sehingga Lambda dapat mengonversi nilai yang dikembalikan ke JSON. Misalnya, U bisa berupa tipe sederhana seperti`String`,`serde_json::Value`, atau struct khusus selama `Serialize` mengimplementasikannya. Ketika kode Anda mencapai pernyataan Ok (U), ini menunjukkan eksekusi yang berhasil, dan fungsi Anda mengembalikan nilai tipeU.
  - Ketika kode Anda menemukan kesalahan (yaitu,`Err(Error)`), fungsi Anda mencatat kesalahan di Amazon CloudWatch dan mengembalikan respons kesalahan tipe`Error`.

Dalam contoh kita, tanda tangan handler terlihat seperti berikut:

```
async fn function_handler(event: LambdaEvent<Value>) -> Result<String, Error>
```

Tanda tangan handler lain yang valid dapat menampilkan yang berikut:

- Menghilangkan `LambdaEvent` pembungkus - Jika Anda menghilangkan`LambdaEvent`, Anda kehilangan akses ke objek konteks Lambda dalam fungsi Anda. Berikut ini adalah contoh dari jenis tanda tangan ini:

  ```
  async fn handler(event: serde_json::Value) -> Result<String, Error>
  ```

- Menggunakan tipe unit sebagai input - Untuk Rust, Anda dapat menggunakan tipe unit untuk mewakili input kosong. Ini biasanya digunakan untuk fungsi dengan pemanggilan berkala dan terjadwal. Berikut ini adalah contoh dari jenis tanda tangan ini:

  ```
  async fn handler(_: ()) -> Result<Value, Error>
  ```

Konvensi penamaan handler

Penangan Lambda di Rust tidak memiliki batasan penamaan yang ketat. Meskipun Anda dapat menggunakan nama apa pun untuk handler Anda, nama fungsi di Rust umumnya masuk`snake_case`.

Untuk aplikasi yang lebih kecil, seperti dalam contoh ini, Anda dapat menggunakan satu `main.rs` file untuk memuat semua kode Anda. Untuk proyek yang lebih besar, `main.rs` harus berisi titik masuk ke fungsi Anda, tetapi Anda dapat memiliki file tambahan untuk itu memisahkan kode Anda menjadi modul logis. Misalnya, Anda mungkin memiliki struktur file berikut:

```
/example-rust
### src/
#   ### main.rs        # Entry point
#   ### handler.rs     # Contains main handler
#   ### services.rs    # [Optional] Back-end service calls
#   ### models.rs      # [Optional] Data models
### Cargo.toml
```

Mendefinisikan dan mengakses objek peristiwa masukan

JSON adalah format input yang paling umum dan standar untuk Lambda Event Functions. Dalam contoh ini, fungsi mengharapkan input yang mirip dengan yang berikut:

```
{
    "order_id": "12345",
    "amount": 199.99,
    "item": "Wireless Headphones"
}
```

Di Rust, Anda dapat menentukan bentuk peristiwa input yang diharapkan dalam struct. Dalam contoh ini, kita mendefinisikan struct berikut untuk mewakili`Order`:

```
#[derive(Deserialize, Serialize)]
struct Order {
    order_id: String,
    amount: f64,
    item: String,
}
```

Struct ini cocok dengan bentuk input yang diharapkan. Dalam contoh ini, `#[derive(Deserialize, Serialize)]` makro secara otomatis menghasilkan kode untuk serialisasi dan deserialisasi. Ini berarti bahwa kita dapat deserialisasi tipe JSON input generik ke dalam struct kita menggunakan metode ini. `serde_json::from_value()` Ini diilustrasikan dalam beberapa baris pertama handler:

```
async fn function_handler(event: LambdaEvent<Value>) -> Result<String, Error> {
    let payload = event.payload;

    // Deserialize the incoming event into Order struct
    let order: Order = serde_json::from_value(payload)?;
    ...
}
```

Anda kemudian dapat mengakses bidang objek. Misalnya, `order.order_id` mengambil nilai dari `order_id` dari input asli.

Pre-defined jenis acara masukan

Ada banyak jenis peristiwa masukan yang telah ditentukan sebelumnya yang tersedia di `aws_lambda_events` peti. Misalnya, jika Anda ingin menjalankan fungsi Anda dengan API Gateway, termasuk impor berikut:

```
use aws_lambda_events::event::apigw::ApiGatewayProxyRequest;
```

Kemudian, pastikan handler utama Anda menggunakan tanda tangan berikut:

```
async fn handler(event: LambdaEvent<ApiGatewayProxyRequest>) -> Result<String, Error> {
    let body = event.payload.body.unwrap_or_default();
    ...
}
```

Lihat peti aws_lambda_events untuk informasi selengkapnya tentang jenis peristiwa masukan yang telah ditentukan sebelumnya.

Mengakses dan menggunakan objek konteks Lambda

Objek konteks Lambda berisi informasi tentang lingkungan pemanggilan, fungsi, dan eksekusi. Di Rust, `LambdaEvent` pembungkus menyertakan objek konteks. Misalnya, Anda dapat menggunakan objek konteks untuk mengambil ID permintaan dari pemanggilan saat ini dengan kode berikut:

```
async fn function_handler(event: LambdaEvent<Value>) -> Result<String, Error> {
    let request_id = event.context.request_id;
    ...
}
```

Untuk informasi lebih lanjut tentang objek konteks, lihatthe section called “Konteks”.

Menggunakan AWS SDK for Rust di handler Anda

Seringkali, Anda akan menggunakan fungsi Lambda untuk berinteraksi dengan atau membuat pembaruan ke sumber daya lain AWS . Cara termudah untuk berinteraksi dengan sumber daya ini adalah dengan menggunakan file AWS SDK for Rust.

Untuk menambahkan dependensi SDK ke fungsi Anda, tambahkan dependensi di file Anda. `Cargo.toml` Kami sarankan hanya menambahkan pustaka yang Anda butuhkan untuk fungsi Anda.

Dalam contoh kode sebelumnya, kita menggunakan`aws_sdk_s3::Client`. Dalam `Cargo.toml` file, Anda dapat menambahkan ketergantungan ini dengan menambahkan baris berikut di bawah `[dependencies]` bagian:

```
aws-sdk-s3 = "1.78.0"
```

> **Note**
>
> Ini mungkin bukan versi terbaru. Pilih versi yang sesuai untuk aplikasi Anda.

Itu, impor dependensi langsung di kode Anda:

```
use aws_sdk_s3::{Client, primitives::ByteStream};
```

Kode contoh kemudian menginisialisasi klien Amazon S3 sebagai berikut:

```
let config = aws_config::load_defaults(aws_config::BehaviorVersion::latest()).await;
let s3_client = Client::new(&config);
```

Setelah menginisialisasi klien SDK, Anda dapat menggunakannya untuk berinteraksi dengan layanan lain AWS . Kode contoh memanggil Amazon S3 `PutObject` API dalam fungsi `upload_receipt_to_s3` helper.

Mengakses variabel lingkungan

Dalam kode handler Anda, Anda dapat mereferensikan variabel lingkungan apa pun dengan menggunakan `env::var` metode ini. Dalam contoh ini, kami mereferensikan variabel `RECEIPT_BUCKET` lingkungan yang ditentukan menggunakan baris kode berikut:

```
let bucket_name = env::var("RECEIPT_BUCKET")
    .map_err(|_| "RECEIPT_BUCKET environment variable is not set")?;
```

Menggunakan status bersama

Anda dapat mendeklarasikan variabel bersama yang independen dari kode handler fungsi Lambda Anda. Variabel ini dapat membantu Anda memuat informasi status selamaFase inisialisasi, sebelum fungsi Anda menerima peristiwa apa pun. Misalnya, Anda dapat mengubah kode di halaman ini untuk

menggunakan status bersama saat menginisialisasi klien Amazon S3 dengan memperbarui fungsi dan tanda tangan `main` handler:

```
async fn function_handler(client: &Client, event: LambdaEvent<Value>) -> Result<String,
 Error> {
    ...
    upload_receipt_to_s3(client, &bucket_name, &key, &receipt_content).await?;
    ...
}

...

#[tokio::main]
async fn main() -> Result<(), Error> {
    let shared_config = aws_config::from_env().load().await;
    let client = Client::new(&shared_config);
    let shared_client = &client;
    lambda_runtime::run(service_fn(move |event: LambdaEvent<Request>| async move {
        handler(&shared_client, event).await
    }))
    .await
```

Praktik terbaik kode untuk Fungsi Acara Lambda Rust

Patuhi pedoman dalam daftar berikut untuk menggunakan praktik pengkodean terbaik saat membangun fungsi Lambda Anda:

- Pisahkan handler Lambda dari logika inti Anda. Ini membuatnya lebih mudah untuk membuat fungsi yang lebih dapat diuji unit.
- Minimalkan kompleksitas dependensi Anda. Utamakan memilih kerangka kerja lebih sederhana yang cepat dimuat dalam memulai lingkungan eksekusi.
- Minimalkan ukuran paket penerapan Anda sesuai kebutuhan runtime-nya. Ini mengurangi jumlah waktu yang diperlukan untuk paket penerapan Anda untuk diunduh dan dibongkar sebelum pemanggilan.

Manfaatkan penggunaan kembali lingkungan eksekusi untuk meningkatkan kinerja fungsi Anda. Inisialisasi klien SDK dan koneksi basis data di luar fungsi handler, dan lakukan caching aset statis secara lokal di direktori `/tmp`. Invokasi selanjutnya yang diproses oleh instans yang sama dari fungsi Anda dapat menggunakan kembali sumber daya ini. Ini menghemat biaya dengan mengurangi waktu pengoperasian fungsi.

Untuk menghindari potensi kebocoran data di seluruh invokasi, jangan menggunakan lingkungan eksekusi untuk menyimpan data pengguna, peristiwa, atau informasi lainnya implikasi keamanan. Jika fungsi Anda bergantung pada status yang dapat disenyapkan yang tidak dapat disimpan dalam memori di dalam handler, pertimbangkan untuk membuat fungsi terpisah atau versi terpisah dari fungsi untuk setiap pengguna.

Gunakan arahan keep-alive untuk mempertahankan koneksi yang persisten. Lambda membersihkan koneksi idle dari waktu ke waktu. Mencoba menggunakan ulang koneksi idle saat mengidentifikasi suatu fungsi akan menyebabkan kesalahan koneksi. Untuk mempertahankan koneksi yang persisten, gunakan arahan tetap aktif yang berkaitan dengan runtime Anda. Sebagai contoh, lihat Reusing Connections with Keep-Alive in Node.js.

Gunakan variabel lingkungan untuk meneruskan parameter operasional ke fungsi Anda. Misalnya, jika Anda ingin menulis ke bucket Amazon S3 alih-alih melakukan hard-coding nama bucket yang Anda tulis, konfigurasikan nama bucket sebagai variabel lingkungan.

Hindari menggunakan pemanggilan rekursif dalam fungsi Lambda Anda, di mana fungsi memanggil dirinya sendiri atau memulai proses yang dapat memanggil fungsi lagi. Hal ini dapat menyebabkan volume invokasi fungsi yang tidak diinginkan dan peningkatan biaya. Jika Anda melihat volume pemanggilan yang tidak diinginkan, setel konkurensi fungsi cadangan untuk `0` segera membatasi semua pemanggilan ke fungsi, saat Anda memperbarui kode.

Jangan gunakan API non-publik yang tidak terdokumentasi dalam kode fungsi Lambda Anda. Untuk runtime AWS Lambda terkelola, Lambda secara berkala menerapkan pembaruan keamanan dan fungsional ke API internal Lambda. Pembaruan API internal ini mungkin tidak kompatibel ke belakang, yang menyebabkan konsekuensi yang tidak diinginkan seperti kegagalan pemanggilan jika fungsi Anda memiliki ketergantungan pada API non-publik ini. Lihat referensi API untuk daftar API yang tersedia untuk umum.

Tulis kode idempoten. Menulis kode idempoten untuk fungsi Anda memastikan bahwa peristiwa duplikat ditangani dengan cara yang sama. Kode Anda harus memvalidasi peristiwa dengan benar dan menangani peristiwa duplikat dengan anggun. Untuk informasi selengkapnya, lihat Bagaimana cara membuat fungsi Lambda saya idempoten? .

Menggunakan objek konteks Lambda untuk mengambil informasi Rust Event Function

Ketika Lambda menjalankan Fungsi Acara Anda, ia menambahkan objek konteks ke LambdaEvent yang diterima handler. Objek ini menyediakan properti dengan informasi tentang lingkungan invokasi, fungsi, dan eksekusi.

Properti konteks

- `request_id`: ID AWS permintaan yang dihasilkan oleh layanan Lambda.
- `deadline`: Batas waktu eksekusi untuk pemanggilan saat ini dalam milidetik.
- `invoked_function_arn`: Nama Sumber Daya Amazon (ARN) dari fungsi Lambda yang dipanggil.
- `xray_trace_id`: ID AWS X-Ray jejak untuk pemanggilan saat ini.
- `client_content`: Objek konteks klien yang dikirim oleh SDK AWS seluler. Bidang ini kosong kecuali fungsi dipanggil menggunakan SDK AWS seluler.
- `identity`: Identitas Amazon Cognito yang memanggil fungsi. Kolom ini kosong kecuali permintaan pemanggilan ke API Lambda dibuat menggunakan AWS kredensil yang dikeluarkan oleh kumpulan identitas Amazon Cognito.
- `env_config`: Konfigurasi fungsi Lambda dari variabel lingkungan lokal. Properti ini mencakup informasi seperti nama fungsi, alokasi memori, versi, dan aliran log.

Mengakses informasi konteks aktif

Fungsi Lambda memiliki akses ke metadata tentang lingkungan mereka dan permintaan invokasi. `LambaEvent`Objek yang diterima oleh penangan fungsi Anda mencakup `context` metadata:

```
use lambda_runtime::{service_fn, LambdaEvent, Error};
use serde_json::{json, Value};

async fn handler(event: LambdaEvent<Value>) -> Result<Value, Error> {
    let invoked_function_arn = event.context.invoked_function_arn;
    Ok(json!({ "message": format!("Hello, this is function
 {invoked_function_arn}!") }))
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    lambda_runtime::run(service_fn(handler)).await
```

```
}
```

Memproses peristiwa HTTP dengan Rust

API Amazon API Gateway, Application Load Balancers, dan URL fungsi Lambda dapat mengirim peristiwa HTTP ke Lambda. Anda dapat menggunakan peti aws_lambda_events dari crates.io untuk memproses peristiwa dari sumber-sumber ini.

Example— Menangani permintaan proxy API Gateway

Perhatikan hal-hal berikut:

- `use aws_lambda_events::apigw::{ApiGatewayProxyRequest, ApiGatewayProxyResponse}`: Peti aws_lambda_events mencakup banyak peristiwa Lambda. Untuk mengurangi waktu kompilasi, gunakan flag fitur untuk mengaktifkan acara yang Anda butuhkan. Contoh:`aws_lambda_events = { version = "0.8.3", default-features = false, features = ["apigw"] }`.
- `use http::HeaderMap`: Impor ini mengharuskan Anda untuk menambahkan peti http ke dependensi Anda.

```
use aws_lambda_events::apigw::{ApiGatewayProxyRequest, ApiGatewayProxyResponse};
use http::HeaderMap;
use lambda_runtime::{service_fn, Error, LambdaEvent};

async fn handler(
    _event: LambdaEvent<ApiGatewayProxyRequest>,
) -> Result<ApiGatewayProxyResponse, Error> {
    let mut headers = HeaderMap::new();
    headers.insert("content-type", "text/html".parse().unwrap());
    let resp = ApiGatewayProxyResponse {
        status_code: 200,
        multi_value_headers: headers.clone(),
        is_base64_encoded: false,
        body: Some("Hello AWS Lambda HTTP request".into()),
        headers,
    };
    Ok(resp)
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    lambda_runtime::run(service_fn(handler)).await
```

```
}
```

Klien runtime Rust untuk Lambda juga menyediakan abstraksi atas jenis acara ini yang dapat Anda kerjakan dengan tipe HTTP asli, terlepas dari layanan mana yang mengirimkan peristiwa. Kode berikut ini setara dengan contoh sebelumnya, dan berfungsi di luar kotak dengan URL fungsi Lambda, Application Load Balancers, dan API Gateway.

> **Note**
>
> Peti lambda_http menggunakan peti lambda_runtime di bawahnya. Anda tidak perlu mengimpor `lambda_runtime` secara terpisah.

Example— Menangani permintaan HTTP

```
use lambda_http::{service_fn, Error, IntoResponse, Request, RequestExt, Response};

async fn handler(event: Request) -> Result<impl IntoResponse, Error> {
    let resp = Response::builder()
        .status(200)
        .header("content-type", "text/html")
        .body("Hello AWS Lambda HTTP request")
        .map_err(Box::new)?;
    Ok(resp)
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    lambda_http::run(service_fn(handler)).await
}
```

Untuk contoh lain tentang cara menggunakan`lambda_http`, lihat contoh kode http-axum pada repositori Labs. AWS GitHub

Contoh acara HTTP Lambda untuk Rust

- Peristiwa HTTP Lambda: Fungsi Rust yang menangani peristiwa HTTP.
- Peristiwa Lambda HTTP dengan header CORS: Fungsi Rust yang menggunakan Tower untuk menyuntikkan header CORS.

- Peristiwa HTTP Lambda dengan sumber daya bersama: Fungsi Rust yang menggunakan sumber daya bersama yang diinisialisasi sebelum penangan fungsi dibuat.

Terapkan Fungsi Peristiwa Rust Lambda dengan arsip file.zip

Halaman ini menjelaskan cara mengkompilasi Rust Event Function Anda, dan kemudian menerapkan biner yang dikompilasi ke AWS Lambda memakai Kargo Lambda. Ini juga menunjukkan cara menerapkan biner yang dikompilasi dengan AWS Command Line Interface dan AWS Serverless Application Model CLI.

Bagian-bagian

- Prasyarat
- Membangun fungsi Rust di macOS, Windows, atau Linux
- Menyebarkan biner fungsi Rust dengan Cargo Lambda
- Memanggil fungsi Rust Anda dengan Cargo Lambda

Prasyarat

- Karat
- AWS CLI versi 2

Membangun fungsi Rust di macOS, Windows, atau Linux

Langkah-langkah berikut menunjukkan cara membuat proyek untuk fungsi Lambda pertama Anda dengan Rust dan mengkompilasinya dengan Kargo Lambda, ekstensi sumber terbuka pihak ketiga untuk alat baris perintah Cargo yang menyederhanakan pembuatan dan penerapan fungsi Rust Lambda.

1. Instal Kargo Lambda, ekstensi sumber terbuka pihak ketiga untuk alat baris perintah Cargo yang menyederhanakan pembuatan dan penerapan fungsi Rust Lambda:

   ```
   cargo install cargo-lambda
   ```

   Untuk opsi instalasi lainnya, lihat Instalasi dalam dokumentasi Cargo Lambda.

2. Buat struktur paket. Perintah ini membuat beberapa kode fungsi dasar di `src/main.rs`. Anda dapat menggunakan kode ini untuk pengujian atau menggantinya dengan milik Anda sendiri.

   ```
   cargo lambda new my-function
   ```

3. Di dalam direktori root paket, jalankan membangun subperintah untuk mengkompilasi kode dalam fungsi Anda.

```
cargo lambda build --release
```

(Opsional) Jika Anda ingin menggunakan AWS Graviton pada Lambda, tambahkan `--arm64` flag untuk mengkompilasi kode Anda untuk CPU ARM.

```
cargo lambda build --release --arm64
```

4. Sebelum menerapkan fungsi Rust Anda, konfigurasikan AWS kredensibilitas pada mesin Anda.

```
aws configure
```

Menyebarkan biner fungsi Rust dengan Cargo Lambda

Gunakan menyebarkan subperintah untuk menyebarkan biner yang dikompilasi ke Lambda. Perintah ini menciptakan sebuah peran eksekusi dan kemudian membuat fungsi Lambda. Untuk menentukan peran eksekusi yang ada, gunakan --bendera peran iam.

```
cargo lambda deploy my-function
```

Menyebarkan biner fungsi Rust Anda dengan AWS CLI

Anda juga dapat menerapkan biner Anda dengan AWS CLI.

1. Gunakan membangun subperintah untuk membangun paket penyebaran.zip.

```
cargo lambda build --release --output-format zip
```

2. Untuk menerapkan paket.zip ke Lambda, jalankan buat-fungsi perintah.

   - Untuk `--runtime`, tentukan `provided.al2023`. Ini adalah OS-only runtime. OS-only runtime digunakan untuk menerapkan binari terkompilasi dan runtime khusus ke Lambda.
   - Untuk `--role`, tentukan ARN dari peran eksekusi.

```
aws lambda create-function \
     --function-name my-function \
```

```
    --runtime provided.al2023 \
    --role arn:aws:iam::111122223333:role/lambda-role \
    --handler rust.handler \
    --zip-file fileb://target/lambda/my-function/bootstrap.zip
```

Menyebarkan biner fungsi Rust Anda dengan AWS SAM CLI

Anda juga dapat menerapkan biner Anda dengan AWS SAM CLI.

1. Buat AWS SAM template dengan definisi sumber daya dan properti. Untuk `Runtime`, tentukan `provided.al2023`. Ini adalah OS-only runtime. OS-only runtime digunakan untuk menerapkan binari terkompilasi dan runtime khusus ke Lambda.

   Untuk informasi selengkapnya tentang menerapkan fungsi Lambda menggunakan AWS SAM, lihat AWS::Serverless::Fungsi di AWS Serverless Application Model Panduan Pengembang.

   Example Sumber daya SAM dan definisi properti untuk biner Rust

   ```
   AWSTemplateFormatVersion: '2010-09-09'
   Transform: AWS::Serverless-2016-10-31
   Description: SAM template for Rust binaries
   Resources:
     RustFunction:
       Type: AWS::Serverless::Function
       Properties:
         CodeUri: target/lambda/my-function/
         Handler: rust.handler
         Runtime: provided.al2023
   Outputs:
     RustFunction:
       Description: "Lambda Function ARN"
       Value: !GetAtt RustFunction.Arn
   ```

2. Gunakan membangun subperintah untuk mengkompilasi fungsi.

   ```
   cargo lambda build --release
   ```

3. Gunakan sam menyebarkan perintah untuk menyebarkan fungsi ke Lambda.

   ```
   sam deploy --guided
   ```

Untuk informasi lebih lanjut tentang membangun fungsi Rust dengan AWS SAM CLI, lihat Membangun fungsi Rust Lambda dengan Cargo Lambda di AWS Serverless Application Model Panduan Pengembang.

Memanggil fungsi Rust Anda dengan Cargo Lambda

Gunakan meminta subperintah untuk menguji fungsi Anda dengan payload.

```
cargo lambda invoke --remote --data-ascii '{"command": "Hello world"}' my-function
```

Memanggil fungsi Rust Anda dengan AWS CLI

Anda juga dapat menggunakan AWS CLI untuk memanggil fungsi.

```
aws lambda invoke --function-name my-function --cli-binary-format raw-in-base64-out --
payload '{"command": "Hello world"}' /tmp/out.txt
```

cli-binary-format opsi diperlukan jika Anda menggunakan AWS CLI versi 2. Untuk menjadikan ini pengaturan default, jalankan `aws configure set cli-binary-format raw-in-base64-out`. Untuk informasi lebih lanjut, lihat AWS CLI opsi baris perintah global yang didukung di AWS Command Line Interface Panduan Pengguna untuk Versi 2.

Bekerja dengan lapisan untuk Fungsi Acara Lambda Rust

Kami tidak menyarankan menggunakan lapisan untuk mengelola dependensi untuk Fungsi Peristiwa Lambda yang ditulis dalam Rust. Ini karena fungsi Rust dikompilasi menjadi satu executable, yang Anda berikan ke Lambda saat Anda menerapkan fungsi Anda. Executable ini berisi kode fungsi dikompilasi Anda, bersama dengan semua dependensinya. Menggunakan lapisan tidak hanya mempersulit proses ini, tetapi juga menyebabkan peningkatan waktu mulai dingin karena fungsi Anda perlu memuat rakitan ekstra secara manual ke dalam memori selama fase init.

Untuk menggunakan dependensi eksternal dengan penangan Rust Anda, sertakan langsung ke dalam paket penerapan Anda. Dengan demikian, Anda menyederhanakan proses penerapan dan juga memanfaatkan pengoptimalan kompiler Rust bawaan. Untuk contoh cara mengimpor dan menggunakan dependensi seperti AWS SDK for Rust dalam fungsi Anda, lihat. the section called “Handler”

Log dan pantau Fungsi Acara Rust Lambda

AWS Lambda secara otomatis memonitor Fungsi Acara Lambda atas nama Anda dan mengirim log ke Amazon. CloudWatch Fungsi Lambda Anda dilengkapi dengan grup CloudWatch log Log dan aliran log untuk setiap instance fungsi Anda. Lingkungan runtime Lambda mengirimkan detail tentang setiap invokasi ke pengaliran log, dan menyampaikan log serta output lain dari kode fungsi Anda.

Untuk informasi selengkapnya, lihat Mengirim log Fungsi Peristiwa Lambda ke Log CloudWatch. Untuk informasi tentang mengonfigurasi format log, lihatMengkonfigurasi JSON dan format log teks biasa. Halaman ini menjelaskan cara menghasilkan output log dari kode fungsi Lambda Anda.

Membuat fungsi yang menulis log

Untuk mengeluarkan log dari kode fungsi Anda, Anda dapat menggunakan fungsi logging apa pun yang menulis ke `stdout` ataustderr, seperti `println!` makro. Contoh berikut digunakan `println!` untuk mencetak pesan ketika fungsi handler dimulai dan sebelum selesai.

```
use lambda_runtime::{service_fn, LambdaEvent, Error};
use serde_json::{json, Value};
async fn handler(event: LambdaEvent<Value>) -> Result<Value, Error> {
    println!("Rust function invoked");
    let payload = event.payload;
    let first_name = payload["firstName"].as_str().unwrap_or("world");
    println!("Rust function responds to {}", &first_name);
    Ok(json!({ "message": format!("Hello, {first_name}!") }))
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    lambda_runtime::run(service_fn(handler)).await
}
```

Menerapkan logging lanjutan dengan peti Tracing

Tracing adalah kerangka kerja untuk menginstrumentasi program Rust untuk mengumpulkan informasi diagnostik berbasis peristiwa yang terstruktur. Kerangka kerja ini menyediakan utilitas untuk menyesuaikan tingkat dan format keluaran logging, seperti membuat pesan log JSON terstruktur. Untuk menggunakan framework ini, Anda harus menginisialisasi `subscriber` sebelum mengimplementasikan fungsi handler. Kemudian, Anda dapat menggunakan melacak makro sepertidebug,`info`, dan`error`, untuk menentukan tingkat logging yang Anda inginkan untuk setiap skenario.

Example— Menggunakan peti Tracing

Perhatikan hal-hal berikut:

- `tracing_subscriber::fmt().json()`: Ketika opsi ini disertakan, log diformat dalam JSON. Untuk menggunakan opsi ini, Anda harus menyertakan `json` fitur dalam `tracing-subscriber` ketergantungan (misalnya,`tracing-subscriber = { version = "0.3.11", features = ["json"] }`).
- `#[tracing::instrument(skip(event), fields(req_id = %event.context.request_id))]`: Anotasi ini menghasilkan rentang setiap kali handler dipanggil. Rentang menambahkan ID permintaan ke setiap baris log.
- `{ %first_name }`: Konstruksi ini menambahkan `first_name` bidang ke baris log tempat ia digunakan. Nilai untuk bidang ini sesuai dengan variabel dengan nama yang sama.

```
use lambda_runtime::{service_fn, Error, LambdaEvent};
use serde_json::{json, Value};
#[tracing::instrument(skip(event), fields(req_id = %event.context.request_id))]
async fn handler(event: LambdaEvent<Value>) -> Result<Value, Error> {
    tracing::info!("Rust function invoked");
    let payload = event.payload;
    let first_name = payload["firstName"].as_str().unwrap_or("world");
    tracing::info!({ %first_name }, "Rust function responds to event");
    Ok(json!({ "message": format!("Hello, {first_name}!") }))
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt().json()
        .with_max_level(tracing::Level::INFO)
        // this needs to be set to remove duplicated information in the log.
        .with_current_span(false)
        // this needs to be set to false, otherwise ANSI color codes will
        // show up in a confusing manner in CloudWatch logs.
        .with_ansi(false)
        // disabling time is handy because CloudWatch will add the ingestion time.
        .without_time()
        // remove the name of the function from every log entry
        .with_target(false)
        .init();
    lambda_runtime::run(service_fn(handler)).await
```

```
}
```

Ketika fungsi Rust ini dipanggil, ia mencetak dua baris log yang mirip dengan yang berikut ini:

```
{"level":"INFO","fields":{"message":"Rust function invoked"},"spans":
[{"req_id":"45daaaa7-1a72-470c-9a62-e79860044bb5","name":"handler"}]}
{"level":"INFO","fields":{"message":"Rust function responds to
 event","first_name":"David"},"spans":[{"req_id":"45daaaa7-1a72-470c-9a62-
e79860044bb5","name":"handler"}]}
```

Runtime Fungsi Peristiwa Lambda

AWS Lambda Fungsi Event (sebelumnya dikenal sebagai fungsi Lambda) mendukung banyak bahasa melalui penggunaan waktu proses. Runtime menyediakan lingkungan khusus bahasa yang menyampaikan peristiwa pemanggilan, informasi konteks, dan respons antara Lambda dan fungsi. Anda dapat menggunakan runtime yang disediakan Lambda, atau membangun sendiri. Untuk informasi tentang runtime sumber terbuka, lihat the section called “Repositori sumber terbuka”. Untuk runtime yang digunakan Lambda Web Functions, lihat the section called “Runtime”.

Lambda adalah agnostik terhadap pilihan runtime Anda. Untuk fungsi sederhana, bahasa yang ditafsirkan seperti Python dan Node.js menawarkan kinerja tercepat. Untuk fungsi dengan komputasi yang lebih kompleks, bahasa yang dikompilasi seperti Java seringkali lebih lambat untuk diinisialisasi tetapi berjalan dengan cepat di handler Lambda. Pilihan runtime juga dipengaruhi oleh preferensi pengembang dan keakraban bahasa.

Setiap rilis bahasa pemrograman utama memiliki runtime terpisah, dengan yang unik pengidentifikasi runtime, seperti `nodejs24.x` atau `python3.14`. Untuk mengkonfigurasi fungsi untuk menggunakan versi bahasa utama baru, Anda perlu mengubah pengidentifikasi runtime. Sejak AWS Lambda tidak dapat menjamin kompatibilitas mundur antara versi utama, ini adalah operasi yang digerakkan oleh pelanggan.

Untuk fungsi didefinisikan sebagai gambar wadah, Anda memilih runtime dan distribusi Linux saat Anda membuat image container. Untuk mengubah runtime, Anda membuat gambar kontainer baru.

Ketika Anda menggunakan arsip file .zip untuk paket deployment, Anda memilih runtime ketika Anda membuat fungsi. Untuk mengubah runtime, Anda dapat memperbarui konfigurasi fungsi. Runtime dipasangkan dengan salah satu distribusi Amazon Linux. Lingkungan eksekusi yang mendasari menyediakan pustaka dan variabel lingkungan tambahan yang dapat Anda akses dari kode fungsi Anda.

Lambda memanggil fungsi Anda dalam lingkungan eksekusi. Lingkungan eksekusi menyediakan lingkungan runtime yang aman dan terisolasi yang mengelola sumber daya yang diperlukan untuk menjalankan fungsi Anda. Lambda menggunakan kembali lingkungan eksekusi dari pemanggilan sebelumnya jika tersedia, atau dapat membuat lingkungan eksekusi baru.

Untuk menggunakan bahasa lain di Lambda, seperti Pergi atau Karat, gunakan OS-only runtime. Area pelaksanaan Lambda menyediakan antarmuka runtime atas adanya event invokasi dan mengirimkan tanggapan. Anda dapat menerapkan bahasa lain dengan menerapkan runtime khusus di samping kode fungsi Anda, atau dalam lapisan.

Waktu aktif yang didukung

Tabel berikut mencantumkan runtime Lambda yang didukung untuk Fungsi Peristiwa dan tanggal penghentian yang diproyeksikan. Setelah runtime tidak digunakan lagi, Anda masih dapat membuat dan memperbarui fungsi untuk jangka waktu terbatas. Untuk informasi selengkapnya, lihat the section called “Penggunaan runtime setelah penghentian”. Tabel ini memberikan tanggal yang diperkirakan saat ini untuk penghentian waktu proses, berdasarkan the section called “Kebijakan penghentian waktu proses”. Tanggal-tanggal ini disediakan untuk tujuan perencanaan dan dapat berubah.

> **Note**
>
> Yang Node.js Runtime 26 dan Python 3.15 sudah masuk pratinjau publik. Runtime pratinjau tidak tercakup oleh SLA Lambda atau Dukungan Teknis, dan tidak boleh digunakan untuk beban kerja produksi.
> Tujuan dari runtime pratinjau adalah untuk mengumpulkan umpan balik dari pelanggan, mitra, dan komunitas bahasa hulu sebelum Ketersediaan Umum, sementara perubahan yang merusak masih dimungkinkan. Anda dapat memberikan umpan balik melalui khusus GitHub masalah untuk Node.js 26 dan Python 3.15. Untuk informasi lebih lanjut, lihat ini posting blog.

> **Important**
>
> Amazon Linux 2 dijadwalkan berakhir pada 30 Juni 2026. Runtime Lambda dan gambar basis kontainer untuk Java 8 (AL2), Java 11 (AL2), Java 17 (AL2), Python 3.10, Python 3.11, dan disediakan. al2 akan terus menerima patch untuk kritis dan terpilih penting Masalah keamanan Amazon Linux 2, selain patch runtime bahasa, hingga tanggal penghentian yang ditunjukkan pada tabel di bawah ini.

Kami menyarankan pelanggan meningkatkan ke runtime berbasis Amazon Linux 2023 sesegera mungkin. Untuk pelanggan yang memutakhirkan ke Java 21 atau Java 25, Anda dapat menggunakan AWS Transform kebiasaan untuk membantu dengan peningkatan ini. Bagi pelanggan yang tidak dapat meningkatkan versi Java mereka, kami menyediakan runtime berbasis Amazon Linux 2023 untuk Java 8, Java 11, dan Java 17 yang mengikuti jadwal penghentian AL2023.

Nama	Pengidentifikasi	Sistem operasi	Tanggal pengusangan	Blokir fungsi buat	Pembaruan fungsi blok
Node.js 26	`nodejs26.x`	Amazon Linux 2023	Tidak dijadwalkan	Tidak dijadwalkan	Tidak dijadwalkan
Node.js 24	`nodejs24.x`	Amazon Linux 2023	30 April 2028	1 Juni 2028	1 Juli 2028
Node.js 22	`nodejs22.x`	Amazon Linux 2023	30 April 2027	29 Juli 2027	31 Agu 2027
Python 3.15	`python3.15`	Amazon Linux 2023	Tidak dijadwalkan	Tidak dijadwalkan	Tidak dijadwalkan
Python 3.14	`python3.14`	Amazon Linux 2023	30 Juni 2029	31 Juli 2029	31 Agu 2029
Python 3.13	`python3.13`	Amazon Linux 2023	30 Juni 2029	31 Juli 2029	31 Agu 2029
Python 3.12	`python3.12`	Amazon Linux 2023	31 Okt 2028	30 November 2028	10 Jan 2029
Python 3.11	`python3.11`	Amazon Linux 2	30 Juni 2027	29 Juli 2027	31 Agu 2027
Python 3.10	`python3.10`	Amazon Linux 2	31 Okt 2026	29 Juli 2027	31 Agu 2027

Nama	Pengidentifikasi	Sistem operasi	Tanggal pengusangan	Blokir fungsi buat	Pembaruan fungsi blok
Jawa 25	`java25`	Amazon Linux 2023	30 Juni 2029	31 Juli 2029	31 Agu 2029
Jawa 21	`java21`	Amazon Linux 2023	30 Juni 2029	31 Juli 2029	31 Agu 2029
Jawa 17	`java17.al2023`	Amazon Linux 2023	30 Juni 2029	31 Juli 2029	31 Agu 2029
Java 11	`java11.al2023`	Amazon Linux 2023	30 Juni 2029	31 Juli 2029	31 Agu 2029
Java 8	`java8.al2023`	Amazon Linux 2023	30 Juni 2029	31 Juli 2029	31 Agu 2029
Jawa 17	`java17`	Amazon Linux 2	30 Juni 2027	29 Juli 2027	31 Agu 2027
Java 11	`java11`	Amazon Linux 2	30 Juni 2027	29 Juli 2027	31 Agu 2027
Java 8	`java8.al2`	Amazon Linux 2	30 Juni 2027	29 Juli 2027	31 Agu 2027
.NET10	`dotnet10`	Amazon Linux 2023	14 November 2028	14 Des 2028	15 Jan 2029
.NET 9 (hanya kontainer)	`dotnet9`	Amazon Linux 2023	10 November 2026	Tidak dijadwalkan	Tidak dijadwalkan
.NET8	`dotnet8`	Amazon Linux 2023	10 November 2026	29 Juli 2027	31 Agu 2027
Ruby 4.0	`ruby4.0`	Amazon Linux 2023	31 Maret 2029	30 April 2029	Mei 31, 2029

Nama	Pengidentifikasi	Sistem operasi	Tanggal pengusangan	Blokir fungsi buat	Pembaruan fungsi blok
Ruby 3.4	`ruby3.4`	Amazon Linux 2023	31 Maret 2028	30 April 2028	Mei 31, 2028
Ruby 3.3	`ruby3.3`	Amazon Linux 2023	31 Maret 2027	29 Juli 2027	31 Agu 2027
OS-only Waktu pelaksanaan	`provided.al2023`	Amazon Linux 2023	30 Juni 2029	31 Juli 2029	31 Agu 2029

> **Note**
>
> Untuk wilayah baru, Lambda tidak akan mendukung runtime yang ditetapkan untuk tidak digunakan lagi dalam 6 bulan ke depan.

Lambda selalu memperbarui runtime terkelola dan gambar basis kontainer yang sesuai dengan tambalan dan dukungan untuk rilis versi minor. Untuk informasi lebih lanjut lihat Pembaruan runtime Lambda.

Untuk berinteraksi secara terprogram dengan orang lain Layanan AWS dan sumber daya dari fungsi Anda, Anda dapat menggunakan salah satu AWS SDK. The Node.js, Python, dan runtime Ruby menyertakan versi AWS SDK. Namun, untuk mempertahankan kontrol penuh atas dependensi Anda, dan untuk memaksimalkan kompatibilitas mundur selama pembaruan runtime otomatis, kami sarankan Anda selalu menyertakan modul SDK yang digunakan kode Anda (bersama dengan dependensi apa pun) dalam paket penerapan fungsi Anda atau dalam Lapisan Lambda.

Sebaiknya gunakan versi SDK yang disertakan runtime hanya jika Anda tidak dapat menyertakan paket tambahan dalam penerapan Anda. Misalnya, saat Anda membuat fungsi menggunakan editor kode konsol Lambda atau menggunakan kode fungsi sebaris dalam CloudFormation templat.

Lambda secara berkala memperbarui versi AWS SDK termasuk dalam Node.js, Python, dan runtime Ruby. Untuk menentukan versi AWS SDK disertakan dalam runtime yang Anda gunakan, lihat bagian berikut:

- Runtime-included Versi SDK (Node.js)
- Runtime-included Versi SDK (Python)
- Runtime-included Versi SDK (Ruby)

Lambda terus mendukung bahasa pemrograman Go setelah penghentian runtime Go 1.x. Untuk informasi lebih lanjut, lihat Bermigrasi AWS Lambda fungsi dari Go1.x runtime ke runtime khusus di Amazon Linux 2 pada AWS Komputasi Blog.

Semua runtime Lambda yang didukung mendukung arsitektur x86_64 dan arm64.

Rilis runtime baru

Lambda menyediakan runtime terkelola untuk versi bahasa baru hanya ketika rilis mencapai fase dukungan jangka panjang (LTS) dari siklus rilis bahasa. Misalnya, untuk Node.js siklus rilis, ketika rilis mencapai fase LTS Aktif.

Sebelum rilis mencapai fase dukungan jangka panjang, itu tetap dalam pengembangan dan masih dapat mengalami perubahan besar. Lambda menerapkan pembaruan runtime secara otomatis secara default, jadi melanggar perubahan pada versi runtime dapat menghentikan fungsi Anda berfungsi seperti yang diharapkan.

Lambda tidak menyediakan runtime terkelola untuk versi bahasa yang tidak dijadwalkan untuk rilis LTS.

Daftar berikut menunjukkan bulan peluncuran target untuk runtime Lambda mendatang. Tanggal-tanggal ini hanya bersifat indikatif dan dapat berubah.

- Node.js 26 - November 2026
- Python 3.15 - November 2026

Kebijakan penghentian waktu proses

Runtime Lambda untuk arsip file.zip dibangun di sekitar kombinasi sistem operasi, bahasa pemrograman, dan pustaka perangkat lunak yang tunduk pada pemeliharaan dan pembaruan keamanan. Kebijakan penghentian standar Lambda adalah menghentikan runtime ketika komponen utama runtime mencapai akhir dukungan jangka panjang komunitas (LTS) dan pembaruan keamanan tidak lagi tersedia. Paling sering, ini adalah runtime bahasa, meskipun dalam beberapa kasus, runtime dapat tidak digunakan lagi karena sistem operasi (OS) mencapai akhir LTS.

Setelah runtime tidak digunakan lagi, AWS mungkin tidak lagi menerapkan patch keamanan atau pembaruan ke runtime tersebut, dan fungsi yang menggunakan runtime tersebut tidak lagi memenuhi syarat untuk dukungan teknis. Runtime yang tidak digunakan lagi disediakan 'apa adanya', tanpa jaminan apa pun, dan mungkin berisi bug, kesalahan, cacat, atau kerentanan lainnya.

Untuk mempelajari selengkapnya tentang mengelola peningkatan runtime dan penghentian, lihat bagian berikut dan Mengelola AWS Lambda peningkatan runtime pada AWS Komputasi Blog.

> **Important**
>
> Lambda terkadang menunda penghentian runtime Lambda untuk jangka waktu terbatas setelah tanggal akhir dukungan versi bahasa yang didukung runtime. Selama periode ini, Lambda hanya menerapkan patch keamanan ke OS runtime. Lambda tidak menerapkan patch keamanan ke runtime bahasa pemrograman setelah mencapai tanggal akhir dukungan.

Model tanggung jawab bersama

Lambda bertanggung jawab untuk mengkurasi dan menerbitkan pembaruan keamanan untuk semua runtime terkelola yang didukung dan gambar basis kontainer. Secara default, Lambda akan menerapkan pembaruan ini secara otomatis ke fungsi menggunakan runtime terkelola. Jika pengaturan pembaruan runtime otomatis default telah diubah, lihat manajemen runtime mengontrol model tanggung jawab bersama. Untuk fungsi yang digunakan menggunakan gambar kontainer, Anda bertanggung jawab untuk membangun kembali image wadah fungsi Anda dari image dasar terbaru dan menyebarkan ulang image kontainer.

Ketika runtime tidak digunakan lagi, tanggung jawab Lambda untuk memperbarui runtime terkelola dan image basis kontainer berhenti. Anda bertanggung jawab untuk meningkatkan fungsi Anda untuk menggunakan runtime atau image dasar yang didukung.

Dalam semua kasus, Anda bertanggung jawab untuk menerapkan pembaruan ke kode fungsi Anda, termasuk dependensinya. Tanggung jawab Anda di bawah model tanggung jawab bersama dirangkum dalam tabel berikut.

Fase siklus proses runtime	Tanggung jawab Lambda	Tanggung jawab Anda
Runtime terkelola yang didukung	Berikan pembaruan runtime reguler dengan patch	Perbarui kode fungsi Anda, termasuk dependensi, untuk

Fase siklus proses runtime	Tanggung jawab Lambda	Tanggung jawab Anda
	keamanan dan pembaruan lainnya. Terapkan pembaruan runtime secara otomatis secara default (lihat the section called “Mode pembaruan runtime” untuk perilaku non-default).	mengatasi kerentanan keamanan apa pun.
Gambar kontainer yang didukung	Berikan pembaruan rutin ke image basis kontainer dengan patch keamanan dan pembaruan lainnya.	Perbarui kode fungsi Anda, termasuk dependensi, untuk mengatasi kerentanan keamanan apa pun. Secara teratur membangun kembali dan menyebarkan kembali image kontainer Anda menggunakan image dasar terbaru.
Runtime terkelola mendekati penghentian	Beri tahu pelanggan sebelum penghentian runtime melalui dokumentasi, Dasbor Health, email, dan Trusted Advisor. Tanggung jawab untuk pembaruan runtime berakhir pada penghentian.	Memantau dokumentasi Lambda, Dasbor Health, email, atau Trusted Advisor untuk informasi penghentian runtime. Tingkatkan fungsi ke runtime yang didukung sebelum runtime sebelumnya tidak digunakan lagi.

Fase siklus proses runtime	Tanggung jawab Lambda	Tanggung jawab Anda
Gambar kontainer mendekati penghentian	Pemberitahuan penghentian tidak tersedia untuk fungsi yang menggunakan gambar kontainer. Tanggung jawab untuk pembaruan gambar basis kontainer berakhir saat penghentian.	Waspadai jadwal penghenti an dan tingkatkan fungsi ke gambar dasar yang didukung sebelum gambar sebelumnya tidak digunakan lagi.

Penggunaan runtime setelah penghentian

Setelah runtime tidak digunakan lagi, AWS mungkin tidak lagi menerapkan patch keamanan atau pembaruan ke runtime tersebut, dan fungsi yang menggunakan runtime tersebut tidak lagi memenuhi syarat untuk dukungan teknis. Meskipun Anda dapat terus memanggil fungsi Anda tanpa batas waktu, AWS sangat merekomendasikan migrasi ke runtime yang didukung. Runtime yang tidak digunakan lagi disediakan 'apa adanya', tanpa jaminan apa pun, dan mungkin berisi bug, kesalahan, cacat, atau kerentanan lainnya. Fungsi yang menggunakan runtime yang tidak digunakan lagi mungkin juga mengalami penurunan kinerja atau masalah lain, seperti kedaluwarsa sertifikat, yang dapat menyebabkan fungsi tersebut berhenti berfungsi dengan baik.

Anda dapat memperbarui fungsi untuk menggunakan runtime yang didukung yang lebih baru kapan saja setelah runtime tidak digunakan lagi. Kami menyarankan untuk menguji fungsi Anda dengan runtime baru sebelum menerapkan perubahan di lingkungan produksi. Anda tidak akan dapat kembali ke runtime yang tidak digunakan lagi setelah pembaruan fungsi diblokir. Kami merekomendasikan menggunakan fungsi versi dan alias untuk mengaktifkan penyebaran yang aman dengan rollback.

Garis waktu berikut menjelaskan apa yang terjadi ketika runtime tidak digunakan lagi:

Fase siklus proses runtime	Saat	Apa
Periode pemberita huan penghentian	Setidaknya 180 hari sebelum penghentian	• AWS mengirimkan pemberitahuan melalui email dan Dasbor Health ke akun yang

Fase siklus proses runtime	Saat	Apa
		memiliki fungsi menggunakan runtime ini di akun mereka `$LATEST` versi. • Fungsi yang terpengaruh juga tercantum dalam Dasbor Health Tab perubahan terjadwal dan AWS Trusted Advisor pemeriksaan runtime yang tidak digunakan lagi.
Penghentian	Tanggal pengusangan	• AWS mungkin tidak lagi menerapkan pembaruan keamanan atau pembaruan lainnya. • Fungsi tidak lagi memenuhi syarat untuk dukungan teknis. • Anda tidak dapat lagi membuat atau memperbarui fungsi menggunakan runtime yang tidak digunakan lagi di konsol Lambda. Anda dapat terus membuat dan memperbarui fungsi melalui AWS CLI, AWS SAM, atau CloudFormation.
Blokir fungsi buat	Setidaknya 30 hari setelah penghentian	• Lambda mulai memblokir pembuatan fungsi baru. • Anda dapat terus memperbarui kode dan konfigurasi untuk fungsi yang ada melalui AWS CLI, AWS SAM, atau CloudFormation..
Pembaruan fungsi blok	Setidaknya 60 hari setelah penghentian	• Lambda mulai memblokir pembaruan kode dan konfigurasi untuk fungsi yang ada. • Anda masih dapat memutakhirkan konfigurasi fungsi ke runtime yang didukung. Namun, memutar kembali ke runtime yang tidak digunakan lagi mungkin diblokir.

> **Note**
>
> Untuk beberapa runtime, AWS menunda tanggal block-fungsi-buat dan block-fungsi-pembaruan di luar 30 dan 60 hari biasa setelah penghentian. AWS telah membuat perubahan ini sebagai tanggapan terhadap umpan balik pelanggan untuk memberi Anda lebih banyak waktu untuk meningkatkan fungsi Anda. Lihat tabel di the section called “Waktu aktif yang didukung” dan the section called “Waktu pengoperasian terdepresiasi” untuk melihat tanggal runtime Anda. Lambda tidak akan mulai memblokir fungsi pembuatan atau pembaruan sebelum tanggal yang diberikan dalam tabel ini.

Menerima pemberitahuan penghentian runtime

Ketika runtime mendekati tanggal penghentiannya, Lambda mengirimi Anda peringatan email jika ada fungsi di Akun AWS Gunakan runtime tersebut. Pemberitahuan juga ditampilkan di Dasbor Health dan di AWS Trusted Advisor.

- Menerima pemberitahuan email:

  Lambda mengirimi Anda peringatan email setidaknya 180 hari sebelum runtime tidak digunakan lagi. Email ini mencantumkan versi $LATEST dari semua fungsi yang menggunakan runtime. Untuk melihat daftar lengkap versi fungsi yang terpengaruh, gunakan Trusted Advisor atau lihat the section called “Dapatkan data tentang fungsi berdasarkan runtime”.

  Lambda mengirimkan pemberitahuan email ke Anda Akun AWS kontak akun utama. Untuk informasi tentang melihat atau memperbarui alamat email di akun Anda, lihat Memperbarui informasi kontak di AWS Referensi Umum.

- Menerima pemberitahuan melalui Dasbor Health:

  The Dasbor Health menampilkan pemberitahuan setidaknya 180 hari sebelum runtime tidak digunakan lagi. Pemberitahuan muncul di Kesehatan akun Anda halaman di bawah Pemberitahuan lainnya. Sumber daya yang terpengaruh tab notifikasi mencantumkan versi $LATEST dari semua fungsi menggunakan runtime.

> **Note**
>
> Untuk melihat daftar lengkap dan terbaru dari versi fungsi yang terpengaruh, gunakan Trusted Advisor atau lihat the section called “Dapatkan data tentang fungsi berdasarkan runtime”.

Dasbor Health notifikasi kedaluwarsa 90 hari setelah runtime yang terpengaruh tidak digunakan lagi.

- Menggunakan AWS Trusted Advisor

  Trusted Advisor menampilkan pemberitahuan setidaknya 180 hari sebelum runtime tidak digunakan lagi. Pemberitahuan muncul di Keamanan halaman. Daftar fungsi Anda yang terpengaruh ditampilkan di bawah AWS Lambda Fungsi Menggunakan Runtime yang Dusang. Daftar fungsi ini menampilkan versi $LATEST dan yang diterbitkan dan diperbarui secara otomatis untuk mencerminkan status fungsi Anda saat ini.

  Anda dapat mengaktifkan notifikasi email mingguan dari Trusted Advisor di Preferensi halaman dari Trusted Advisor konsol.

Waktu pengoperasian terdepresiasi

Runtime berikut telah mencapai akhir dukungan:

Nama	Pengidentifikasi	Sistem operasi	Tanggal pengusangan	Fungsi blok buat	Pembaruan fungsi blok
OS-only Waktu pelaksanaan	`provided.al2`	Amazon Linux 2	31 Juli 2026	29 Juli 2027	31 Agu 2027
Node.js 20	`nodejs20.x`	Amazon Linux 2023	30 April 2026	29 Juli 2027	31 Agu 2027
Ruby 3.2	`ruby3.2`	Amazon Linux 2	31 Maret 2026	29 Juli 2027	31 Agu 2027

Nama	Pengidentifikasi	Sistem operasi	Tanggal pengusangan	Fungsi blok buat	Pembaruan fungsi blok
Python 3.9	`python3.9`	Amazon Linux 2	15 Des 2025	29 Juli 2027	31 Agu 2027
Node.js 18	`nodejs18.x`	Amazon Linux 2	1 September 2025	29 Juli 2027	31 Agu 2027
.NET6	`dotnet6`	Amazon Linux 2	20 Des 2024	29 Juli 2027	31 Agu 2027
Python 3.8	`python3.8`	Amazon Linux 2	14 Okt 2024	29 Juli 2027	31 Agu 2027
Node.js 16	`nodejs16.x`	Amazon Linux 2	12 Juni 2024	29 Juli 2027	31 Agu 2027
.NET 7 (hanya kontainer)	`dotnet7`	Amazon Linux 2	Mei 14, 2024	N/A	N/A
Java 8	`java8`	Amazon Linux	8 Jan 2024	8 Februari 2024	31 Agu 2027
Jalankan 1.x	`go1.x`	Amazon Linux	8 Jan 2024	8 Februari 2024	31 Agu 2027
OS-only Waktu pelaksanaan	`provided`	Amazon Linux	8 Jan 2024	8 Februari 2024	31 Agu 2027
Ruby 2.7	`ruby2.7`	Amazon Linux 2	7 Des 2023	9 Jan 2024	31 Agu 2027
Node.js 14	`nodejs14.x`	Amazon Linux 2	4 Des 2023	9 Jan 2024	31 Agu 2027

Nama	Pengidentifikasi	Sistem operasi	Tanggal pengusangan	Fungsi blok buat	Pembaruan fungsi blok
Python 3.7	`python3.7`	Amazon Linux	4 Des 2023	9 Jan 2024	31 Agu 2027
.NET Core 3.1	`dotnetcore3.1`	Amazon Linux 2	3 Apr 2023	3 Apr 2023	3 Mei 2023
Node.js 12	`nodejs12.x`	Amazon Linux 2	31 Maret 2023	31 Maret 2023	30 Apr 2023
Python 3.6	`python3.6`	Amazon Linux	18 Juli 2022	18 Juli 2022	29 Agustus 2022
.NET 5 (hanya kontainer)	`dotnet5.0`	Amazon Linux 2	10 Mei 2022	N/A	N/A
.NET Core 2.1	`dotnetcore2.1`	Amazon Linux	5 Jan 2022	5 Jan 2022	13 Apr 2022
Node.js 10	`nodejs10.x`	Amazon Linux 2	30 Juli 2021	30 Juli 2021	14 Februari 2022
Ruby 2.5	`ruby2.5`	Amazon Linux	30 Juli 2021	30 Juli 2021	31 Maret 2022
Python 2.7	`python2.7`	Amazon Linux	15 Juli 2021	15 Juli 2021	Mei 30, 2022
Node.js 8.10	`nodejs8.10`	Amazon Linux	6 Maret 2020	4 Februari 2020	6 Maret 2020
Node.js 4.3	`nodejs4.3`	Amazon Linux	5 Maret 2020	3 Februari 2020	5 Maret 2020
Node.js 4.3 tepi	`nodejs4.3-edge`	Amazon Linux	5 Maret 2020	31 Maret 2019	30 April 2019

Nama	Pengidentifikasi	Sistem operasi	Tanggal pengusangan	Fungsi blok buat	Pembaruan fungsi blok
Node.js 6.10	`nodejs6.10`	Amazon Linux	12 Agustus 2019	12 Juli 2019	12 Agustus 2019
.NET Core 1.0	`dotnetcore1.0`	Amazon Linux	27 Juni 2019	30 Juni 2019	30 Juli 2019
.NET Core 2.0	`dotnetcore2.0`	Amazon Linux	30 Mei 2019	30 April 2019	30 Mei 2019
Node.js 0.10	`nodejs`	Amazon Linux	30 Agustus 2016	30 September 2016	31 Okt 2016

Dalam hampir semua kasus, tanggal akhir penggunaan versi bahasa atau sistem operasi diketahui pada jauh-jauh hari sebelumnya. Tautan berikut memberikan jadwal akhir masa pakai untuk setiap bahasa yang didukung Lambda sebagai runtime terkelola.

Kebijakan dukungan bahasa dan framework

- Node.js -- github.com
- Python – devguide.python.org
- Ruby – www.ruby-lang.org
- Java – www.oracle.com and FAQ Corretto
- Go – golang.org
- .NET -- dotnet.microsoft.com

Memahami bagaimana Lambda mengelola pembaruan versi runtime

Lambda selalu memperbarui setiap runtime yang dikelola dengan pembaruan keamanan, perbaikan bug, fitur baru, peningkatan kinerja, dan dukungan untuk rilis versi minor. Pembaruan runtime ini diterbitkan sebagai versi runtime. Lambda menerapkan pembaruan runtime ke Fungsi Peristiwa dengan memigrasikan fungsi dari versi runtime sebelumnya ke versi runtime baru.

Secara default, untuk fungsi yang menggunakan runtime terkelola, Lambda menerapkan pembaruan runtime secara otomatis. Dengan pembaruan runtime otomatis, Lambda menanggung beban operasional untuk menambal versi runtime. Bagi sebagian besar pelanggan, pembaruan otomatis adalah pilihan yang tepat. Anda dapat mengubah perilaku default ini dengan mengonfigurasi pengaturan manajemen runtime.

Lambda juga menerbitkan setiap versi runtime baru sebagai gambar kontainer. Untuk memperbarui versi runtime untuk fungsi berbasis container, Anda harus membuat image container baru dari image dasar yang diperbarui dan menerapkan ulang fungsi Anda.

Setiap versi runtime dikaitkan dengan nomor versi dan ARN (Amazon Resource Name). Nomor versi runtime menggunakan skema penomoran yang didefinisikan Lambda, terlepas dari nomor versi yang digunakan bahasa pemrograman. Nomor versi runtime tidak selalu berurutan. Misalnya, versi 42 mungkin diikuti oleh versi 45. ARN versi runtime adalah pengenal unik untuk setiap versi runtime. Anda dapat melihat ARN versi runtime fungsi Anda saat ini di konsol Lambda, atau `INIT_START`baris log fungsi Anda. Untuk informasi tentang mengelola pembaruan runtime dalam aplikasi yang penting bagi kesehatan, lihat. the section called “Izin”

Versi runtime tidak boleh disamakan dengan pengidentifikasi runtime. Setiap runtime memiliki pengenal runtime yang unik, seperti atau. `python3.14` `nodejs24.x` Ini sesuai dengan setiap rilis bahasa pemrograman utama. Versi runtime menjelaskan versi patch dari runtime individual.

> **Note**
>
> ARN untuk nomor versi runtime yang sama dapat bervariasi antara Wilayah AWS dan arsitektur CPU.

Topik

- Kompatibilitas mundur
- Mode pembaruan runtime

- Two-phase peluncuran versi runtime
- Mengkonfigurasi pengaturan manajemen runtime Lambda
- Mengembalikan versi runtime Lambda
- Mengidentifikasi perubahan versi runtime Lambda
- Memahami model tanggung jawab bersama untuk manajemen runtime Lambda
- Mengontrol izin pembaruan runtime Lambda untuk aplikasi dengan kepatuhan tinggi

Kompatibilitas mundur

Lambda berusaha untuk menyediakan pembaruan runtime yang kompatibel dengan fungsi yang ada. Namun, seperti halnya patching perangkat lunak, ada kasus yang jarang terjadi di mana pembaruan runtime dapat berdampak negatif pada fungsi yang ada. Misalnya, patch keamanan dapat mengekspos masalah mendasar dengan fungsi yang ada yang bergantung pada perilaku tidak aman sebelumnya.

Saat membangun dan menerapkan fungsi Anda, penting untuk memahami cara mengelola dependensi Anda untuk menghindari potensi ketidakcocokan dengan pembaruan runtime masa depan. Misalnya, fungsi Anda memiliki ketergantungan pada paket A, yang pada gilirannya bergantung pada paket B. Kedua paket disertakan dalam runtime Lambda (misalnya, mereka bisa menjadi bagian dari SDK atau dependensinya, atau bagian dari pustaka sistem runtime).

Pertimbangkan skenario berikut:

Deployment	Patching kompatibel	Alasan
• Package A: Gunakan dari runtime • Package B: Gunakan dari runtime	Ya	Pembaruan runtime di masa mendatang untuk paket A dan B kompatibel ke belakang.
• Package A: Dalam paket penyebaran • Package B: Dalam paket penyebaran	Ya	Penerapan Anda diutamakan, sehingga pembaruan runtime future untuk paket A dan B tidak berpengaruh.

Deployment	Patching kompatibel	Alasan
• Package A: Dalam paket penyebaran • Package B: Gunakan dari runtime	Ya*	Pembaruan runtime di masa mendatang ke paket B kompatibel ke belakang. * Jika A dan B digabungkan dengan erat, masalah kompatibilitas dapat terjadi. Misalnya, `botocore` paket `boto3` dan dalam AWS SDK untuk Python harus digunakan bersama.
• Package A: Gunakan dari runtime • Package B: Dalam paket penyebaran	Tidak	Pembaruan runtime di masa mendatang ke paket A mungkin memerlukan versi terbaru dari paket B. Namun, versi paket B yang diterapkan lebih diutamakan, dan mungkin tidak kompatibel dengan versi terbaru dari paket A.

Untuk menjaga kompatibilitas dengan pembaruan runtime di masa mendatang, ikuti praktik terbaik berikut:

- Jika memungkinkan, paketkan semua dependensi: Sertakan semua pustaka yang diperlukan, termasuk AWS SDK dan dependensinya, dalam paket penerapan Anda. Ini memastikan serangkaian komponen yang stabil dan kompatibel.
- Gunakan SDK yang disediakan runtime dengan hemat: Hanya mengandalkan SDK yang disediakan runtime jika Anda tidak dapat menyertakan paket tambahan (misalnya, saat menggunakan editor kode konsol Lambda atau kode sebaris dalam templat). AWS CloudFormation
- Hindari mengganti pustaka sistem: Jangan gunakan pustaka sistem operasi khusus yang mungkin bertentangan dengan pembaruan runtime masa depan.

Mode pembaruan runtime

Lambda berusaha untuk menyediakan pembaruan runtime yang kompatibel dengan fungsi yang ada. Namun, seperti halnya patching perangkat lunak, ada kasus yang jarang terjadi di mana pembaruan runtime dapat berdampak negatif pada fungsi yang ada. Misalnya, patch keamanan dapat mengekspos masalah mendasar dengan fungsi yang ada yang bergantung pada perilaku tidak aman sebelumnya. Kontrol manajemen runtime Lambda membantu mengurangi risiko dampak pada beban kerja Anda jika terjadi ketidakcocokan versi runtime yang jarang terjadi. Untuk setiap versi fungsi ($LATESTatau versi yang diterbitkan), Anda dapat memilih salah satu mode pembaruan runtime berikut:

- Otomatis (default) - Secara otomatis memperbarui ke versi runtime terbaru dan aman menggunakanTwo-phase peluncuran versi runtime. Kami merekomendasikan mode ini untuk sebagian besar pelanggan sehingga Anda selalu mendapat manfaat dari pembaruan runtime.
- Pembaruan fungsi - Perbarui ke versi runtime terbaru dan aman saat Anda memperbarui fungsi Anda. Saat memperbarui fungsi, Lambda memperbarui runtime fungsi Anda ke versi runtime terbaru dan aman. Pendekatan ini menyinkronkan pembaruan runtime dengan penerapan fungsi, memberi Anda kendali atas kapan Lambda menerapkan pembaruan runtime. Dengan mode ini, Anda dapat mendeteksi dan mengurangi ketidakcocokan pembaruan runtime yang jarang terjadi lebih awal. Saat menggunakan mode ini, Anda harus memperbarui fungsi secara teratur agar runtime tetap up to date.
- Manual - Perbarui versi runtime Anda secara manual. Anda menentukan versi runtime dalam konfigurasi fungsi Anda. Fungsi ini menggunakan versi runtime ini tanpa batas waktu.

  Dalam kasus yang jarang terjadi di mana versi runtime baru tidak kompatibel dengan fungsi yang ada, Anda dapat menggunakan mode ini untuk memutar kembali fungsi Anda ke versi runtime sebelumnya. Sebaiknya jangan menggunakan mode Manual untuk mencoba mencapai konsistensi runtime di seluruh penerapan. Untuk informasi selengkapnya, lihat Mengembalikan versi runtime Lambda.

Tanggung jawab untuk menerapkan pembaruan runtime ke fungsi Anda bervariasi sesuai dengan mode pembaruan runtime yang Anda pilih. Untuk informasi selengkapnya, lihat Memahami model tanggung jawab bersama untuk manajemen runtime Lambda.

Two-phase peluncuran versi runtime

Lambda memperkenalkan versi runtime baru dengan urutan sebagai berikut:

1. Pada fase pertama, Lambda menerapkan versi runtime baru setiap kali Anda membuat atau memperbarui fungsi. Sebuah fungsi akan diperbarui saat Anda memanggil operasi UpdateFunctionCodeatau UpdateFunctionConfigurationAPI.
2. Pada fase kedua, Lambda memperbarui fungsi apa pun yang menggunakan mode pembaruan runtime Otomatis dan yang belum diperbarui ke versi runtime yang baru.

Durasi keseluruhan proses peluncuran bervariasi sesuai dengan beberapa faktor, termasuk tingkat keparahan patch keamanan apa pun yang termasuk dalam pembaruan runtime.

Jika Anda secara aktif mengembangkan dan menerapkan fungsi Anda, kemungkinan besar Anda akan mengambil versi runtime baru selama fase pertama. Ini menyinkronkan pembaruan runtime dengan pembaruan fungsi. Jika versi runtime terbaru berdampak negatif pada aplikasi Anda, pendekatan ini memberi Anda kesempatan untuk mengambil tindakan korektif yang cepat. Fungsi yang tidak dalam pengembangan aktif masih menerima manfaat operasional dari pembaruan runtime otomatis selama fase kedua.

Pendekatan ini tidak memengaruhi fungsi yang disetel ke pembaruan Fungsi atau mode Manual. Fungsi yang menggunakan mode pembaruan Fungsi menerima pembaruan runtime terbaru hanya ketika Anda membuat atau memperbaruinya. Fungsi yang menggunakan mode Manual tidak menerima pembaruan runtime.

Lambda menerbitkan versi runtime baru secara bertahap dan bergulir. Wilayah AWS Jika fungsi Anda disetel ke mode pembaruan Otomatis atau Fungsi, kemungkinan fungsi yang diterapkan secara bersamaan ke Wilayah yang berbeda, atau pada waktu yang berbeda di Wilayah yang sama, akan mengambil versi runtime yang berbeda. Pelanggan yang memerlukan konsistensi versi runtime yang terjamin di seluruh lingkungan mereka harus menggunakan gambar kontainer untuk menerapkan fungsinya. Mode Manual dirancang sebagai mitigasi sementara untuk mengaktifkan rollback versi runtime jika versi runtime tidak kompatibel dengan fungsi Anda.

Mengkonfigurasi pengaturan manajemen runtime Lambda

Anda dapat mengonfigurasi pengaturan manajemen runtime untuk Fungsi Peristiwa Lambda menggunakan konsol Lambda atau (). AWS Command Line Interface AWS CLI

> **Note**
>
> Anda dapat mengonfigurasi pengaturan manajemen runtime secara terpisah untuk setiap versi fungsi.

Untuk mengonfigurasi cara Lambda memperbarui versi runtime Anda (konsol)

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih nama Fungsi Acara.
3. Pada tab Kode, di bawah Pengaturan runtime, pilih Edit konfigurasi manajemen runtime.
4. Di bawah Konfigurasi manajemen runtime, pilih salah satu dari berikut ini:

   - Agar fungsi Anda diperbarui ke versi runtime terbaru secara otomatis, pilih Otomatis.
   - Agar fungsi Anda diperbarui ke versi runtime terbaru saat Anda mengubah fungsi, pilih Pembaruan fungsi.
   - Agar fungsi Anda diperbarui ke versi runtime terbaru hanya ketika Anda mengubah ARN versi runtime, pilih Manual. Anda dapat menemukan ARN versi runtime di bawah konfigurasi manajemen Runtime. Anda juga dapat menemukan ARN di `INIT_START` baris log fungsi Anda.

   Untuk informasi selengkapnya tentang opsi ini, lihat Mode pembaruan waktu proses.
5. Pilih Simpan.

Untuk mengonfigurasi cara Lambda memperbarui versi runtime Anda ()AWS CLI

Untuk mengonfigurasi manajemen runtime untuk suatu fungsi, jalankan perintah put-runtime-management-config AWS CLI . Saat menggunakan `Manual` mode, Anda juga harus menyediakan ARN versi runtime.

```
aws lambda put-runtime-management-config \
  --function-name my-function \
  --update-runtime-on Manual \
  --runtime-version-arn arn:aws:lambda:us-
east-2::runtime:8eeff65f6809a3ce81507fe733fe09b835899b99481ba22fd75b5a7338290ec1
```

Anda akan melihat output yang serupa dengan yang berikut:

```
{
  "UpdateRuntimeOn": "Manual",
  "FunctionArn": "arn:aws:lambda:us-east-2:111122223333:function:my-function",
  "RuntimeVersionArn": "arn:aws:lambda:us-
east-2::runtime:8eeff65f6809a3ce81507fe733fe09b835899b99481ba22fd75b5a7338290ec1"
```

```
}
```

Mengembalikan versi runtime Lambda

Jika versi runtime baru tidak kompatibel dengan Fungsi Acara yang sudah ada, Anda dapat memutar kembali versi runtime ke versi sebelumnya. Ini membuat aplikasi Anda tetap berfungsi dan meminimalkan gangguan, memberikan waktu untuk memperbaiki ketidakcocokan sebelum kembali ke versi runtime terbaru.

Lambda tidak memaksakan batas waktu berapa lama Anda dapat menggunakan versi runtime tertentu. Namun, kami sangat menyarankan untuk memperbarui ke versi runtime terbaru sesegera mungkin untuk mendapatkan manfaat dari patch keamanan terbaru, peningkatan kinerja, dan fitur. Lambda menyediakan opsi untuk memutar kembali ke versi runtime sebelumnya hanya sebagai mitigasi sementara jika terjadi masalah kompatibilitas pembaruan runtime yang jarang terjadi. Fungsi yang menggunakan versi runtime sebelumnya untuk jangka waktu yang lama mungkin pada akhirnya mengalami penurunan kinerja atau masalah, seperti kedaluwarsa sertifikat, yang dapat menyebabkan mereka berhenti bekerja dengan benar.

Anda dapat memutar kembali versi runtime dengan cara berikut:

- Menggunakan mode pembaruan runtime Manual
- Menggunakan versi fungsi yang diterbitkan

Untuk informasi selengkapnya, lihat Memperkenalkan kontrol manajemen AWS Lambda runtime di Blog AWS Compute.

Kembalikan versi runtime menggunakan mode pembaruan runtime Manual

Jika Anda menggunakan mode pembaruan versi runtime otomatis, atau menggunakan versi **$LATEST** runtime, Anda dapat memutar kembali versi runtime menggunakan mode Manual. Untuk versi fungsi yang ingin Anda putar kembali, ubah mode pembaruan versi runtime ke Manual dan tentukan ARN dari versi runtime sebelumnya. Untuk informasi selengkapnya tentang menemukan ARN dari versi runtime sebelumnya, lihat. Mengidentifikasi perubahan versi runtime Lambda

> **Note**
>
> Jika `$LATEST` versi fungsi Anda dikonfigurasi untuk menggunakan mode Manual, maka Anda tidak dapat mengubah arsitektur CPU atau versi runtime yang digunakan fungsi Anda. Untuk membuat perubahan ini, Anda harus mengubah ke mode pembaruan Otomatis atau Fungsi.

Kembalikan versi runtime menggunakan versi fungsi yang diterbitkan

Versi fungsi yang diterbitkan adalah snapshot yang tidak dapat diubah dari kode `$LATEST` fungsi dan konfigurasi pada saat Anda membuatnya. Dalam mode Otomatis, Lambda secara otomatis memperbarui versi runtime dari versi fungsi yang diterbitkan selama fase dua peluncuran versi runtime. Dalam mode pembaruan Fungsi, Lambda tidak memperbarui versi runtime dari versi fungsi yang diterbitkan.

Versi fungsi yang diterbitkan menggunakan mode pembaruan Fungsi karenanya membuat snapshot statis dari kode fungsi, konfigurasi, dan versi runtime. Dengan menggunakan mode pembaruan Fungsi dengan versi fungsi, Anda dapat menyinkronkan pembaruan runtime dengan penerapan Anda. Anda juga dapat mengoordinasikan rollback versi kode, konfigurasi, dan runtime dengan mengarahkan lalu lintas ke versi fungsi yang diterbitkan sebelumnya.

Anda dapat mengintegrasikan pendekatan ini ke dalam integrasi berkelanjutan dan pengiriman berkelanjutan (CI/CD) untuk rollback otomatis sepenuhnya jika terjadi ketidakcocokan pembaruan runtime yang jarang terjadi. Saat menggunakan pendekatan ini, Anda harus memperbarui fungsi Anda secara teratur dan menerbitkan versi fungsi baru untuk mengambil pembaruan runtime terbaru. Lihat informasi yang lebih lengkap di Memahami model tanggung jawab bersama untuk manajemen runtime Lambda.

Mengidentifikasi perubahan versi runtime Lambda

Nomor versi runtime dan ARN dicatat di `INIT_START` baris log, yang dipancarkan Lambda CloudWatch ke Log setiap kali menciptakan lingkungan eksekusi baru. Karena lingkungan eksekusi menggunakan versi runtime yang sama untuk semua pemanggilan fungsi, Lambda memancarkan `INIT_START` baris log hanya ketika Lambda mengeksekusi fase init. Lambda tidak memancarkan baris log ini untuk setiap pemanggilan fungsi. Lambda memancarkan baris log ke CloudWatch Log, tetapi tidak terlihat di konsol.

> **Note**
>
> Nomor versi runtime tidak selalu berurutan. Misalnya, versi 42 mungkin diikuti oleh versi 45.

Example Contoh baris log INIT_START

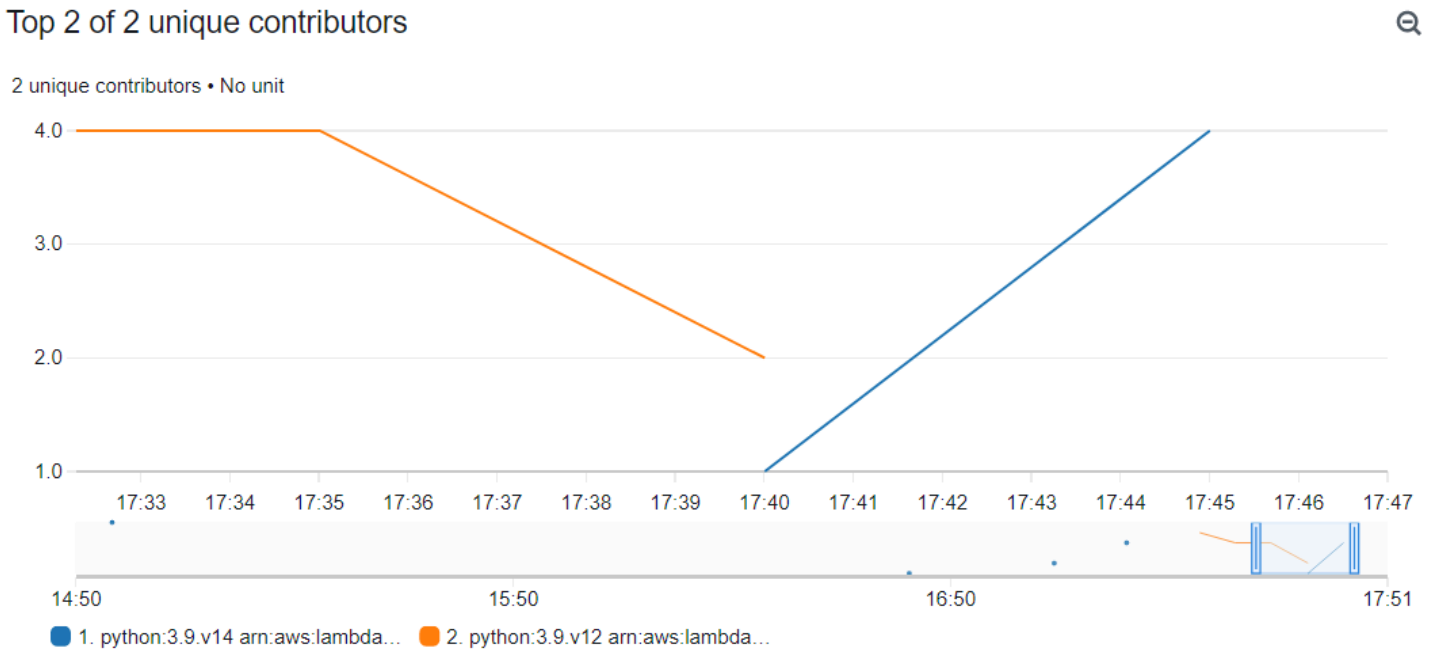
```
INIT_START Runtime Version: python:3.13.v14    Runtime Version ARN: arn:aws:lambda:eu-
south-1::runtime:7b620fc2e66107a1046b140b9d320295811af3ad5d4c6a011fad1fa65127e9e6I
```

Daripada bekerja secara langsung dengan log, Anda dapat menggunakan Amazon CloudWatch Contributor Insights untuk mengidentifikasi transisi antar versi runtime. Aturan berikut menghitung versi runtime yang berbeda dari setiap baris `INIT_START` log. Untuk menggunakan aturan, ganti contoh nama grup log `/aws/lambda/*` dengan awalan yang sesuai untuk fungsi atau grup fungsi Anda.

```
{
  "Schema": {
    "Name": "CloudWatchLogRule",
    "Version": 1
  },
  "AggregateOn": "Count",
  "Contribution": {
    "Filters": [
      {
        "Match": "eventType",
        "In": [
          "INIT_START"
        ]
      }
    ],
    "Keys": [
      "runtimeVersion",
      "runtimeVersionArn"
    ]
  },
  "LogFormat": "CLF",
  "LogGroupNames": [
    "/aws/lambda/*"
  ],
  "Fields": {
```

```
    "1": "eventType",
    "4": "runtimeVersion",
    "8": "runtimeVersionArn"
  }
}
```

Laporan CloudWatch Contributor Insights berikut menunjukkan contoh transisi versi runtime seperti yang ditangkap oleh aturan sebelumnya. Baris oranye menunjukkan inisialisasi lingkungan eksekusi untuk versi runtime sebelumnya (python: 3.13.v12), dan garis biru menunjukkan inisialisasi lingkungan eksekusi untuk versi runtime baru (python: 3.13.v14).



Memahami model tanggung jawab bersama untuk manajemen runtime Lambda

Lambda bertanggung jawab untuk mengkurasi dan menerbitkan pembaruan keamanan untuk semua runtime terkelola dan gambar kontainer yang didukung. Tanggung jawab untuk memperbarui Fungsi Acara yang ada untuk menggunakan versi runtime terbaru bervariasi tergantung pada mode pembaruan runtime yang Anda gunakan.

Lambda bertanggung jawab untuk menerapkan pembaruan runtime ke semua fungsi yang dikonfigurasi untuk menggunakan mode Pembaruan runtime Otomatis.

Untuk fungsi yang dikonfigurasi dengan mode pembaruan runtime pembaruan Fungsi, Anda bertanggung jawab untuk memperbarui fungsi secara teratur. Lambda bertanggung jawab untuk menerapkan pembaruan runtime saat Anda melakukan pembaruan tersebut. Jika Anda tidak

memperbarui fungsi Anda, maka Lambda tidak memperbarui runtime. Jika Anda tidak memperbarui fungsi Anda secara teratur, maka kami sangat menyarankan untuk mengonfigurasinya untuk pembaruan runtime otomatis sehingga terus menerima pembaruan keamanan.

Untuk fungsi yang dikonfigurasi untuk menggunakan mode pembaruan runtime Manual, Anda bertanggung jawab untuk memperbarui fungsi agar menggunakan versi runtime terbaru. Kami sangat menyarankan Anda menggunakan mode ini hanya untuk memutar kembali versi runtime sebagai mitigasi sementara jika terjadi ketidakcocokan pembaruan runtime yang jarang terjadi. Kami juga menyarankan agar Anda mengubah ke mode Otomatis secepat mungkin untuk meminimalkan waktu di mana fungsi Anda tidak ditambal.

Jika Anda menggunakan gambar kontainer untuk menerapkan fungsi Anda, maka Lambda bertanggung jawab untuk menerbitkan gambar dasar yang diperbarui. Dalam hal ini, Anda bertanggung jawab untuk membangun kembali image kontainer fungsi Anda dari image dasar terbaru dan menerapkan kembali image container.

Ini dirangkum dalam tabel berikut:

Mode deployment	Tanggung jawab Lambda	Tanggung jawab pelanggan
Runtime terkelola, mode Otomatis	Publikasikan versi runtime baru yang berisi tambalan terbaru. Terapkan patch runtime ke fungsi yang ada.	Kembalikan ke versi runtime sebelumnya jika terjadi masalah kompatibilitas pembaruan runtime yang jarang terjadi. Ikuti praktik terbaik untuk kompatibilitas mundur.
Runtime terkelola, Mode pembaruan Fungsi	Publikasikan versi runtime baru yang berisi tambalan terbaru.	Perbarui fungsi secara teratur untuk mengambil versi runtime terbaru. Ganti fungsi ke mode Otomatis saat Anda tidak memperbarui fungsi secara teratur. Kembalikan ke versi runtime sebelumnya jika terjadi masalah kompatibilitas pembaruan runtime yang jarang terjadi. Ikuti praktik terbaik untuk kompatibilitas mundur.

Mode deployment	Tanggung jawab Lambda	Tanggung jawab pelanggan
Runtime terkelola, mode Manual	Publikasikan versi runtime baru yang berisi tambalan terbaru.	Gunakan mode ini hanya untuk rollback runtime sementara jika terjadi masalah kompatibilitas pembaruan runtime yang jarang terjadi. Ganti fungsi ke mode pembaruan Otomatis atau Fungsi dan versi runtime terbaru sesegera mungkin.
Gambar kontainer	Publikasikan gambar kontainer baru yang berisi tambalan terbaru.	Menerapkan ulang fungsi secara teratur menggunakan gambar dasar kontainer terbaru untuk mengambil tambalan terbaru.

Untuk informasi selengkapnya tentang tanggung jawab bersama AWS, lihat Model Tanggung Jawab Bersama.

Mengontrol izin pembaruan runtime Lambda untuk aplikasi dengan kepatuhan tinggi

Untuk memenuhi persyaratan penambalan, pelanggan Lambda biasanya mengandalkan pembaruan runtime otomatis. Jika aplikasi Anda tunduk pada persyaratan kesegaran patching yang ketat, Anda mungkin ingin membatasi penggunaan versi runtime sebelumnya. Anda dapat membatasi kontrol manajemen runtime Lambda dengan menggunakan AWS Identity and Access Management (IAM) untuk menolak pengguna di akses AWS akun Anda ke operasi API. PutRuntimeManagementConfig Operasi ini digunakan untuk memilih mode pembaruan runtime untuk Fungsi Acara.

Menolak akses ke operasi ini menyebabkan semua fungsi default ke mode Otomatis. Anda dapat menerapkan pembatasan ini di seluruh organisasi Anda dengan menggunakan kebijakan kontrol layanan (SCP). Jika Anda harus memutar kembali fungsi ke versi runtime yang lebih lama, Anda dapat memberikan pengecualian kebijakan berdasarkan kasus per kasus.

Mengambil data tentang Lambda Event Functions yang menggunakan runtime usang

Saat runtime Lambda mendekati penghentian, Lambda memberi tahu Anda melalui email dan memberikan pemberitahuan di dan. Dasbor Health Trusted Advisor Email dan notifikasi ini mencantumkan versi fungsi $ LATEST menggunakan runtime. Untuk mencantumkan semua versi Fungsi Acara yang menggunakan runtime tertentu, Anda dapat menggunakan AWS Command Line Interface (AWS CLI) atau salah satu AWS SDK.

Jika Anda memiliki sejumlah besar fungsi yang menggunakan runtime yang akan usang, Anda juga dapat menggunakan AWS CLI atau AWS SDK untuk membantu Anda memprioritaskan pembaruan ke fungsi yang paling sering dipanggil.

Lihat bagian berikut untuk mempelajari cara menggunakan AWS CLI dan AWS SDK untuk mengumpulkan data tentang Fungsi Peristiwa yang menggunakan runtime tertentu.

Daftar versi fungsi yang menggunakan runtime tertentu

Untuk mencantumkan semua versi Fungsi Peristiwa Lambda yang menggunakan runtime tertentu, jalankan perintah berikut. AWS CLI Perintah ini tidak mengembalikan Fungsi Web Lambda. Ganti `RUNTIME_IDENTIFIER` dengan nama runtime yang sudah tidak digunakan lagi dan pilih sendiri. Wilayah AWS Untuk daftar hanya versi fungsi $LATEST, hilangkan `--function-version ALL` dari perintah.

```
aws lambda list-functions --function-version ALL --region us-east-1 --output text --
query "Functions[?Runtime=='RUNTIME_IDENTIFIER'].FunctionArn"
```

> **Tip**
>
> Contoh perintah mencantumkan fungsi di `us-east-1` wilayah untuk tertentu Akun AWS Anda harus mengulangi perintah ini untuk setiap wilayah di mana akun Anda memiliki fungsi dan untuk masing-masing Akun AWS.

Anda juga dapat mencantumkan Fungsi Acara yang menggunakan runtime tertentu menggunakan salah satu AWS SDK. Contoh kode berikut menggunakan V3 AWS SDK untuk JavaScript dan AWS SDK untuk Python (Boto3) untuk mengembalikan daftar fungsi ARN untuk fungsi menggunakan

runtime tertentu. Kode contoh juga mengembalikan grup CloudWatch log untuk masing-masing fungsi yang terdaftar. Anda dapat menggunakan grup log ini untuk menemukan tanggal pemanggilan terakhir untuk fungsi tersebut. Lihat bagian berikut the section called “Mengidentifikasi fungsi yang paling umum dan paling baru dipanggil” untuk informasi lebih lanjut.

Node.js

Example JavaScript kode untuk daftar fungsi menggunakan runtime tertentu

```
import { LambdaClient, ListFunctionsCommand } from "@aws-sdk/client-lambda";
const lambdaClient = new LambdaClient();

const command = new ListFunctionsCommand({
    FunctionVersion: "ALL",
    MaxItems: 50
});
const response = await lambdaClient.send(command);

for (const f of response.Functions){
    if (f.Runtime == '<your_runtime>'){ // Use the runtime id, e.g. 'nodejs24.x' or
 'python3.14'
        console.log(f.FunctionArn);
        // get the CloudWatch log group of the function to
        // use later for finding the last invocation date
        console.log(f.LoggingConfig.LogGroup);
    }
}
// If your account has more functions than the specified
// MaxItems, use the returned pagination token in the
// next request with the 'Marker' parameter
if ('NextMarker' in response){
    let paginationToken = response.NextMarker;
  }
```

Python

Example Kode Python untuk mencantumkan fungsi menggunakan runtime tertentu

```
import boto3
from botocore.exceptions import ClientError

def list_lambda_functions(target_runtime):
```

```
    lambda_client = boto3.client('lambda')

    response = lambda_client.list_functions(
        FunctionVersion='ALL',
        MaxItems=50
    )
    if not response['Functions']:
            print("No Lambda functions found")
    else:
        for function in response['Functions']:
            if function['PackageType']=='Zip' and function['Runtime'] ==
 target_runtime:
                print(function['FunctionArn'])
                # Print the CloudWatch log group of the function
                # to use later for finding last invocation date
                print(function['LoggingConfig']['LogGroup'])

    if 'NextMarker' in response:
       pagination_token = response['NextMarker']

if __name__ == "__main__":
    # Replace python3.12 with the appropriate runtime ID for your Lambda functions
    list_lambda_functions('python3.12')
```

Untuk mempelajari lebih lanjut tentang menggunakan AWS SDK untuk mencantumkan fungsi Anda menggunakan ListFunctionstindakan, lihat dokumentasi SDK untuk bahasa pemrograman pilihan Anda.

Anda juga dapat menggunakan fitur Kueri AWS Config lanjutan untuk mencantumkan semua Fungsi Peristiwa Lambda yang menggunakan runtime yang terpengaruh. AWS Config tidak mengembalikan Fungsi Web Lambda. Kueri ini hanya mengembalikan fungsi $LATEST versi, tetapi Anda dapat menggabungkan kueri ke daftar fungsi di semua wilayah dan beberapa Akun AWS dengan satu perintah. Untuk mempelajari selengkapnya, lihat Menanyakan Status AWS Auto Scaling Sumber Daya Konfigurasi Saat Ini di Panduan AWS Config Pengembang.

Mengidentifikasi fungsi yang paling umum dan paling baru dipanggil

Jika Akun AWS berisi fungsi yang menggunakan runtime yang seharusnya tidak digunakan lagi, Anda mungkin ingin memprioritaskan fungsi pembaruan yang sering dipanggil atau fungsi yang baru saja dipanggil.

Jika Anda hanya memiliki beberapa fungsi, Anda dapat menggunakan konsol CloudWatch Log untuk mengumpulkan informasi ini dengan melihat aliran log fungsi Anda. Lihat Melihat data log yang dikirim ke CloudWatch Log untuk informasi selengkapnya.

Untuk melihat jumlah pemanggilan fungsi terbaru, Anda juga dapat menggunakan informasi CloudWatch metrik yang ditampilkan di konsol Lambda. Untuk melihat informasi ini, lakukan hal berikut:

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih fungsi yang ingin Anda lihat statistik pemanggilan.
3. Pilih tab Pantau.
4. Tetapkan periode waktu yang ingin Anda lihat statistik untuk menggunakan pemilih rentang tanggal. Pemanggilan terbaru ditampilkan di panel Pemanggilan.

Untuk akun dengan jumlah fungsi yang lebih besar, akan lebih efisien untuk mengumpulkan data ini secara terprogram menggunakan AWS CLI atau salah satu AWS SDK menggunakan tindakan API DescribeLogStreamsdan GetMetricStatistics.

Contoh berikut menyediakan cuplikan kode menggunakan V3 AWS SDK untuk JavaScript dan AWS SDK untuk Python (Boto3) untuk mengidentifikasi tanggal pemanggilan terakhir untuk fungsi tertentu dan untuk menentukan jumlah pemanggilan untuk fungsi tertentu dalam 14 hari terakhir.

Node.js

Example JavaScript kode untuk menemukan waktu pemanggilan terakhir untuk suatu fungsi

```
import { CloudWatchLogsClient, DescribeLogStreamsCommand } from "@aws-sdk/client-
cloudwatch-logs";
const cloudWatchLogsClient = new CloudWatchLogsClient();
const command = new DescribeLogStreamsCommand({
    logGroupName: '<your_log_group_name>',
    orderBy: 'LastEventTime',
    descending: true,
    limit: 1
});
try {
    const response = await cloudWatchLogsClient.send(command);
    const lastEventTimestamp = response.logStreams.length > 0 ?
        response.logStreams[0].lastEventTimestamp : null;
```

```
    // Convert the UNIX timestamp to a human-readable format for display
    const date = new Date(lastEventTimestamp).toLocaleDateString();
    const time = new Date(lastEventTimestamp).toLocaleTimeString();
    console.log(`${date} ${time}`);

} catch (e){
    console.error('Log group not found.')
}
```

Python

Example Kode Python untuk menemukan waktu pemanggilan terakhir untuk suatu fungsi

```
import boto3
from datetime import datetime

cloudwatch_logs_client  = boto3.client('logs')

response = cloudwatch_logs_client.describe_log_streams(
    logGroupName='<your_log_group_name>',
    orderBy='LastEventTime',
    descending=True,
    limit=1
)

try:
    if len(response['logStreams']) > 0:
        last_event_timestamp = response['logStreams'][0]['lastEventTimestamp']
        print(datetime.fromtimestamp(last_event_timestamp/1000)) # Convert timestamp
 from ms to seconds
    else:
        last_event_timestamp = None
except:
    print('Log group not found')
```

Tip

Anda dapat menemukan nama grup log fungsi Anda menggunakan operasi ListFunctionsAPI. Lihat kode di the section called “Daftar versi fungsi yang menggunakan runtime tertentu” untuk contoh bagaimana melakukan ini.

Node.js

Example JavaScript kode untuk menemukan jumlah pemanggilan dalam 14 hari terakhir

```
import { CloudWatchClient, GetMetricStatisticsCommand } from "@aws-sdk/client-
cloudwatch";
const cloudWatchClient = new CloudWatchClient();
const command = new GetMetricStatisticsCommand({
    Namespace: 'AWS/Lambda',
    MetricName: 'Invocations',
    StartTime: new Date(Date.now()-86400*1000*14), // 14 days ago
    EndTime: new Date(Date.now()),
    Period: 86400 * 14, // 14 days.
    Statistics: ['Sum'],
    Dimensions: [{
        Name: 'FunctionName',
        Value: '<your_function_name>'
    }]
});
const response = await cloudWatchClient.send(command);
const invokesInLast14Days = response.Datapoints.length > 0 ?
    response.Datapoints[0].Sum : 0;

console.log('Number of invocations: ' + invokesInLast14Days);
```

Python

Example Kode Python untuk menemukan jumlah pemanggilan dalam 14 hari terakhir

```
import boto3
from datetime import datetime, timedelta

cloudwatch_client = boto3.client('cloudwatch')

response = cloudwatch_client.get_metric_statistics(
    Namespace='AWS/Lambda',
    MetricName='Invocations',
    Dimensions=[
        {
            'Name': 'FunctionName',
            'Value': '<your_function_name>'
        },
    ],
```

```
    StartTime=datetime.now() - timedelta(days=14),
    EndTime=datetime.now(),
    Period=86400 * 14, # 14 days
    Statistics=[
        'Sum'
    ]
)

if len(response['Datapoints']) > 0:
    invokes_in_last_14_days = int(response['Datapoints'][0]['Sum'])
else:
    invokes_in_last_14_days = 0

print(f'Number of invocations: {invokes_in_last_14_days}')
```

Memodifikasi lingkungan waktu pengoperasian

Anda dapat menggunakan ekstensi internal untuk memodifikasi proses waktu pengoperasian. Ekstensi internal bukan proses terpisah—ekstensi ini berjalan sebagai bagian dari proses waktu pengoperasian.

Lambda menyediakan variabel lingkungan spesifik bahasa yang dapat Anda atur untuk menambahkan opsi dan alat pada waktu pengoperasian. Lambda juga menyediakan skrip pembungkus, yang memungkinkan Lambda mendelegasikan startup waktu pengoperasian ke skrip Anda. Anda dapat membuat skrip pembungkus untuk mengustomisasikan perilaku startup waktu pengoperasian.

Language-specific variabel lingkungan

Lambda mendukung cara konfigurasi saja untuk mengaktifkan kode yang akan dipramuat selama inisialisasi fungsi melalui variabel lingkungan spesifik bahasa berikut:

- `JAVA_TOOL_OPTIONS`— Di Java, Lambda mendukung variabel lingkungan ini untuk mengatur variabel baris perintah tambahan di Lambda. Variabel lingkungan ini memungkinkan Anda menentukan inisialisasi alat, khususnya peluncuran agen bahasa pemrograman asli atau Java menggunakan opsi `agentlib` atau `javaagent`. Untuk informasi selengkapnya, lihat variabel `JAVA_TOOL_OPTIONS` lingkungan.
- `NODE_OPTIONS`— Tersedia dalam Node.js runtime.
- `DOTNET_STARTUP_HOOKS` – Pada NET Core 3.1 dan di atas, variabel lingkungan ini menentukan jalur ke rakitan (dll) yang dapat Lambda gunakan.

Menggunakan variabel lingkungan spesifik bahasa adalah cara yang lebih disukai untuk mengatur properti startup.

Skrip pembungkus

Anda dapat membuat skrip pembungkus untuk menyesuaikan perilaku startup runtime dari Fungsi Peristiwa Lambda Anda. Dengan skrip pembungkus, Anda dapat mengatur parameter konfigurasi yang tidak dapat diatur melalui variabel lingkungan khusus bahasa.

> **Note**
>
> Pemanggilan mungkin gagal jika skrip pembungkus tidak berhasil memulai proses runtime.

Skrip pembungkus didukung pada semua runtime Lambda asli. Skrip pembungkus tidak didukung pada OS-only runtime (keluarga `provided` runtime).

Saat Anda menggunakan skrip pembungkus untuk fungsi, Lambda memulai waktu pengoperasian menggunakan skrip Anda. Lambda mengirim ke skrip Anda jalur ke interpreter dan semua argumen asli untuk startup waktu pengoperasian standar. Skrip Anda dapat memperluas atau mengubah perilaku startup program. Misalnya, skrip dapat menyuntikkan dan mengubah argumen, mengatur variabel lingkungan, atau menangkap metrik, kesalahan, dan informasi diagnostik lainnya.

Anda menentukan skrip dengan mengatur nilai dari variabel lingkungan `AWS_LAMBDA_EXEC_WRAPPER` sebagai jalur sistem file dari biner atau skrip yang dapat dijalankan.

Contoh: Buat dan gunakan skrip pembungkus sebagai lapisan Lambda

Dalam contoh berikut, Anda membuat skrip pembungkus untuk memulai interpreter Python dengan opsi `-X importtime`. Saat Anda menjalankan fungsi, Lambda membuat entri log untuk menampilkan durasi waktu impor untuk tiap impor.

Untuk membuat dan menggunakan skrip pembungkus sebagai lapisan

1. Buat direktori untuk layer:

   ```
   mkdir -p python-wrapper-layer/bin
   cd python-wrapper-layer/bin
   ```

2. Dalam `bin` direktori, paste kode berikut ke file baru bernama`importtime_wrapper`. Ini adalah skrip pembungkus.

   ```
   #!/bin/bash

   # the path to the interpreter and all of the originally intended arguments
   args=("$@")

   # the extra options to pass to the interpreter
   extra_args=("-X" "importtime")

   # insert the extra options
   args=("${args[@]:0:$#-1}" "${extra_args[@]}" "${args[@]: -1}")

   # start the runtime with the extra options
   exec "${args[@]}"
   ```

3. Berikan izin yang dapat dieksekusi skrip:

```
chmod +x importtime_wrapper
```

4. Buat file.zip untuk layer:

```
cd ..
zip -r ../python-wrapper-layer.zip .
```

5. Konfirmasikan bahwa file.zip Anda memiliki struktur direktori berikut:

```
python-wrapper-layer.zip
# bin
    # importtime_wrapper
```

6. Buat layer menggunakan paket.zip.
7. Buat Fungsi Acara menggunakan konsol Lambda.
   a. Buka Konsol Lambda.
   b. Pilih Buat Fungsi Acara.
   c. Masukkan nama Fungsi.
   d. Untuk Runtime, pilih runtime Python terbaru yang didukung.
   e. Pilih Buat Fungsi Acara.
8. Tambahkan lapisan ke fungsi Anda.
   a. Pilih fungsi Anda, lalu pilih tab Kode jika belum dipilih.
   b. Gulir ke bawah ke bagian Layers, lalu pilih Add a layer.
   c. Untuk sumber Layer, pilih Custom layers, dan kemudian pilih layer Anda dari daftar dropdown Custom layers.
   d. Untuk Versi, pilih 1.
   e. Pilih Tambahkan.
9. Tambahkan variabel lingkungan pembungkus.
   a. Pilih tab Konfigurasi, lalu pilih variabel Lingkungan.
   b. Pada Variabel lingkungan, pilih Edit.
   c. Pilih Tambahkan variabel lingkungan.
   d. Untuk Kunci, masukkan `AWS_LAMBDA_EXEC_WRAPPER`.

e. Untuk Nilai, masukkan `/opt/bin/importtime_wrapper` (`/opt/`+ struktur folder lapisan.zip Anda).

f. Pilih Simpan.

10. Uji skrip pembungkus.

a. Pilih tab Uji.

b. Di bawah acara Uji, pilih Uji. Anda tidak perlu membuat acara pengujian—acara default berfungsi.

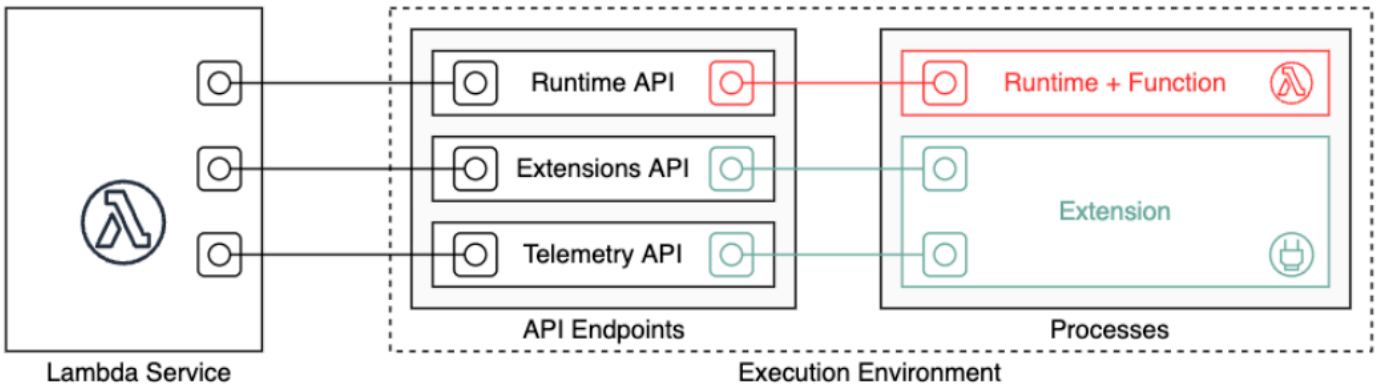
c. Gulir ke bawah ke Keluaran Log. Karena skrip pembungkus Anda memulai interpreter Python dengan opsi `-X importtime`, log menunjukkan waktu yang diperlukan untuk setiap impor. Sebagai contoh:

```
532 |           collections
import time:        63 |         63 |           _functools
import time:      1053 |       3646 |         functools
import time:      2163 |       7499 |       enum
import time:       100 |        100 |         _sre
import time:       446 |        446 |           re._constants
import time:       691 |       1136 |         re._parser
import time:       378 |        378 |         re._casefix
import time:       670 |       2283 |       re._compiler
import time:       416 |        416 |       copyreg
```

Menggunakan API runtime Lambda untuk runtime kustom

AWS Lambda menyediakan API HTTP untuk runtime kustom Fungsi Peristiwa Lambda untuk menerima acara pemanggilan dari Lambda dan mengirim data respons kembali dalam lingkungan eksekusi Lambda. Bagian ini berisi referensi API untuk API runtime Lambda.

> **Instans Terkelola Lambda mendukung permintaan bersamaan**
>
> Instans Terkelola Lambda menggunakan API runtime yang sama dengan fungsi yang menggunakan penyedia kapasitas default Lambda. Perbedaan utamanya adalah Instans Terkelola dapat menerima secara bersamaan `/next` dan `/response` permintaan hingga batas yang dikonfigurasi`AWS_LAMBDA_MAX_CONCURRENCY`. Hal ini memungkinkan beberapa pemanggilan untuk diproses secara bersamaan dalam lingkungan eksekusi tunggal. Untuk informasi selengkapnya tentang Instans Terkelola, lihatMemahami lingkungan eksekusi Instans Terkelola Lambda.



Spesifikasi OpenAPI untuk versi API runtime 2018-06-01 tersedia di runtime-api.zip

Untuk membuat URL permintaan API, runtime mendapatkan titik akhir API dari variabel lingkungan `AWS_LAMBDA_RUNTIME_API`, menambahkan versi API, dan menambahkan jalur sumber daya yang diinginkan.

Example Permintaan

```
curl "http://${AWS_LAMBDA_RUNTIME_API}/2018-06-01/runtime/invocation/next"
```

Metode API

- Invokasi berikutnya
- Respons invokasi
- Kesalahan inisialisasi
- Kesalahan invokasi
- After-Restore (hanya berlaku untuk SnapStart)
- Mengembalikan kesalahan (hanya berlaku untuk SnapStart)

Invokasi berikutnya

Jalur – `/runtime/invocation/next`

Metode – GET

Runtime mengirimkan pesan ini ke Lambda untuk meminta peristiwa invokasi. Isi respons berisi muatan dari invokasi, yang merupakan dokumen JSON yang berisi data kejadian dari pemicu fungsi. Header respons berisi data tambahan tentang invokasi.

Header respons

- `Lambda-Runtime-Aws-Request-Id`— Peristiwa yang memicu pemanggilan fungsi. Sumber peristiwa menyediakan ID permintaan, atau Lambda membuatnya secara otomatis saat menelan. ID permintaan tunggal dapat menghasilkan beberapa upaya pemanggilan. Gunakan di jalur URL saat mengirim respons atau kesalahan.

  Misalnya, `8476a536-e9f4-11e8-9739-2dfe598c3fcd`.
- `Lambda-Runtime-Deadline-Ms` – Tanggal saat fungsi berakhir, dalam milidetik waktu Unix.

  Misalnya, `1542409706888`.
- `Lambda-Runtime-Invoked-Function-Arn` – ARN dari fungsi, versi, atau alias Lambda yang ditentukan dalam invokasi.

  Misalnya, `arn:aws:lambda:us-east-2:123456789012:function:custom-runtime`.
- `Lambda-Runtime-Trace-Id` – Header pelacakan AWS X-Ray.

  Misalnya, `Root=1-5bef4de7-ad49b0e87f6ef6c87fc2e700;Parent=9a9197af755a6419;Sampled=1`.
- `Lambda-Runtime-Client-Context`— Untuk pemanggilan dari AWS Mobile SDK, data tentang aplikasi dan perangkat klien.

- `Lambda-Runtime-Cognito-Identity`— Untuk pemanggilan dari AWS Mobile SDK, data tentang penyedia identitas Amazon Cognito.
- `Lambda-Runtime-Invocation-Id`— Pengidentifikasi unik untuk upaya pemanggilan ini.

Jangan atur waktu habis permintaan `GET` karena respons dapat tertunda. Antara saat Lambda mem-bootstrap runtime dan saat runtime memiliki acara untuk kembali, proses runtime mungkin dibekukan selama beberapa detik.

ID permintaan (`Lambda-Runtime-Aws-Request-Id`) mengidentifikasi peristiwa unik. ID permintaan disediakan oleh sumber peristiwa atau dibuat secara otomatis oleh Lambda saat dikonsumsi. Gunakan ID permintaan di jalur URL saat mengirim respons atau kesalahan.

ID pemanggilan (`Lambda-Runtime-Invocation-Id`) mewakili upaya pemanggilan tunggal untuk suatu peristiwa. ID permintaan tunggal dapat menghasilkan beberapa upaya pemanggilan, masing-masing dengan ID pemanggilan uniknya sendiri. Lambda menggunakan setiap ID pemanggilan tepat sekali dan tidak pernah menggunakannya kembali. Gema nilai ini kembali `/response` dan `/error` panggil. Header bersifat opsional untuk kompatibilitas mundur dengan runtime yang ada—menghilangkannya tidak memicu penolakan. Lambda hanya menolak dengan `400 InvalidInvocationId` ketika header hadir tetapi nilainya tidak cocok dengan pemanggilan aktif.

Header pelacakan berisi ID jejak, ID induk, dan keputusan penyampelan. Jika permintaan diambil sampelnya, permintaan diambil sampelnya oleh Lambda atau layanan hulu. Waktu pengoperasian harus menetapkan `_X_AMZN_TRACE_ID` dengan nilai header. X-Ray SDK membaca ini untuk mendapatkan ID dan menentukan apakah akan melacak permintaan.

Respons invokasi

Jalur – `/runtime/invocation/`*`AwsRequestId`*`/response`

Metode – POST

Setelah fungsi selesai dijalankan, runtime mengirimkan respon invokasi untuk Lambda. Untuk invokasi sinkron, Lambda mengirimkan respons ke klien.

Minta header

`Lambda-Runtime-Invocation-Id`— Gema kembali nilai yang diterima dari`/next`. Lambda menolak permintaan dengan `400 InvalidInvocationId` jika nilainya tidak cocok dengan pemanggilan aktif.

Example permintaan berhasil

```
REQUEST_ID=156cb537-e2d4-11e8-9b34-d36013741fb9
INVOCATION_ID=<value from Lambda-Runtime-Invocation-Id response header>
curl "http://${AWS_LAMBDA_RUNTIME_API}/2018-06-01/runtime/invocation/$REQUEST_ID/
response"  -d "SUCCESS" --header "Lambda-Runtime-Invocation-Id: $INVOCATION_ID"
```

Kesalahan inisialisasi

Jika fungsi mengembalikan kesalahan atau runtime mengalami kesalahan saat inisialisasi, runtime menggunakan metode ini untuk melaporkan kesalahan ke Lambda.

Jalur – `/runtime/init/error`

Metode – POST

Header

`Lambda-Runtime-Function-Error-Type`— Jenis kesalahan yang ditemui runtime.
Header ini opsional. Lambda menerima nilai string apa pun; kami sarankan menggunakan format`<Category.Reason>`, di mana Kategori adalah `Runtime` atau `Function` dan Alasan dimulai dengan huruf besar. Contoh:

- `Runtime.NoSuchHandler`
- `Runtime.APIKeyNotFound`
- `Runtime.ConfigInvalid`
- `Runtime.BeforeSnapshotError`(untuk SnapStart)
- `Runtime.UnknownReason`

Nilai yang tidak cocok dengan pola ini dinormalisasi ke `Runtime.Unknown` atau`Function.Unknown`.

Parameter tubuh

`ErrorRequest` – Informasi tentang kesalahan. Wajib: tidak.

Bidang ini adalah objek JSON dengan struktur berikut:

```
{
      errorMessage: string (text description of the error),
      errorType: string,
```

```
      stackTrace: array of strings
}
```

Perhatikan bahwa Lambda menerima nilai apa pun untuk `errorType`.

Contoh berikut menunjukkan pesan kesalahan fungsi Lambda ketika fungsi tidak dapat mengurai data peristiwa yang disediakan dalam invokasi.

Example Kesalahan fungsi

```
{
      "errorMessage" : "Error parsing event data.",
      "errorType" : "InvalidEventDataException",
      "stackTrace": [ ]
}
```

Parameter isi respons

- `StatusResponse` – String. Informasi status, dikirim dengan kode respons 202.
- `ErrorResponse`— Informasi kesalahan tambahan, dikirim dengan kode respons kesalahan. ErrorResponse berisi jenis kesalahan dan pesan kesalahan.

Kode respons

- 202 – Diterima
- 403 – Dilarang
- 500 - Kesalahan kontainer. Non-recoverable negara. Runtime harus segera keluar.

Example permintaan kesalahan inisialisasi

```
ERROR="{\"errorMessage\" : \"Failed to load function.\", \"errorType\" :
 \"InvalidFunctionException\"}"
curl "http://${AWS_LAMBDA_RUNTIME_API}/2018-06-01/runtime/init/error" -d "$ERROR" --
header "Lambda-Runtime-Function-Error-Type: Unhandled"
```

Kesalahan invokasi

Jika fungsi mengembalikan kesalahan atau runtime mengalami kesalahan, runtime menggunakan metode ini untuk melaporkan kesalahan ke Lambda.

Jalur – `/runtime/invocation/AwsRequestId/error`

Metode – POST

Header

`Lambda-Runtime-Function-Error-Type`— Jenis kesalahan yang ditemui runtime. Wajib: tidak.

Header ini terdiri dari nilai string. Lambda menerima string apa pun, tetapi kami merekomendasikan format <category.reason>. Contoh:

- Runtime.NoSuchHandler
- Runtime.APIKeyNotFound
- Runtime.ConfigInvalid
- Runtime.UnknownReason

`Lambda-Runtime-Invocation-Id`— Gema kembali nilai yang diterima dari`/next`. Lambda menolak permintaan dengan `400 InvalidInvocationId` jika nilainya tidak cocok dengan pemanggilan aktif.

Parameter tubuh

`ErrorRequest` – Informasi tentang kesalahan. Wajib: tidak.

Bidang ini adalah objek JSON dengan struktur berikut:

```
{
      errorMessage: string (text description of the error),
      errorType: string,
      stackTrace: array of strings
}
```

Perhatikan bahwa Lambda menerima nilai apa pun untuk `errorType`.

Contoh berikut menunjukkan pesan kesalahan fungsi Lambda ketika fungsi tidak dapat mengurai data peristiwa yang disediakan dalam invokasi.

Example Kesalahan fungsi

```
{
      "errorMessage" : "Error parsing event data.",
```

```
      "errorType" : "InvalidEventDataException",
      "stackTrace": [ ]
}
```

Parameter isi respons

- `StatusResponse` – String. Informasi status, dikirim dengan kode respons 202.
- `ErrorResponse`— Informasi kesalahan tambahan, dikirim dengan kode respons kesalahan. ErrorResponse berisi jenis kesalahan dan pesan kesalahan.

Kode respons

- 202 – Diterima
- 400 – Permintaan Buruk
- 403 – Dilarang
- 500 - Kesalahan kontainer. Non-recoverable negara. Runtime harus segera keluar.

Example permintaan kesalahan

```
REQUEST_ID=156cb537-e2d4-11e8-9b34-d36013741fb9
ERROR="{\"errorMessage\" : \"Error parsing event data.\", \"errorType\" :
 \"InvalidEventDataException\"}"
curl "http://${AWS_LAMBDA_RUNTIME_API}/2018-06-01/runtime/invocation/$REQUEST_ID/error"
 -d "$ERROR" --header "Lambda-Runtime-Function-Error-Type: Unhandled"
```

After-Restore (hanya berlaku untuk SnapStart)

Jalur – `/runtime/restore/next`

Metode – GET

Setelah kait sebelum-snapshot selesai, runtime memanggil. `GET /runtime/restore/next` Ini adalah panggilan pemblokiran gaya iterator, mirip dengan, yang memberi `/runtime/invocation/next` sinyal ke Lambda bahwa runtime siap untuk lingkungan eksekusi yang akan di-snapshot. Permintaan memblokir hingga Lambda mengembalikan lingkungan eksekusi dari snapshot, lalu mengembalikan respons HTTP 200 dengan badan kosong.

Header

Tidak diperlukan header.

Kode respons

- 200 - Lambda memulihkan lingkungan eksekusi. Jalankan kait setelah pemulihan. Badan respons kosong.
- 403 — Dilarang. Runtime tidak dalam keadaan yang memungkinkan `/restore/next` (misalnya, runtime telah dipanggil `/invocation/next` atau`/restore/next`).
- 404 - tidak SnapStart diaktifkan untuk fungsi ini.
- 500 – Kesalahan penampung. Lingkungan eksekusi berada dalam keadaan tidak dapat dipulihkan. Keluar dari proses runtime.

Sintaksis Permintaan

```
GET /2018-06-01/runtime/restore/next HTTP/1.1
Host: ${AWS_LAMBDA_RUNTIME_API}
```

Sintaksis Respons

```
HTTP/1.1 200 OK
Content-Length: 0
```

> **Note**
>
> Jangan menyetel soket sisi klien atau batas waktu baca pada permintaan Runtime API ini (atau lainnya). Ini adalah panggilan pemblokiran gaya iterator; Lambda membekukan lingkungan eksekusi saat permintaan terbuka. Permintaan dapat tetap terbuka selama masa pakai snapshot (berpotensi berhari-hari, berminggu-minggu, atau lebih lama) tanpa koneksi dianggap menganggur dari layanan Lambda.

Mengembalikan kesalahan (hanya berlaku untuk SnapStart)

Jika hook setelah pemulihan gagal atau runtime mengalami kesalahan selama pemulihan, runtime menggunakan metode ini untuk melaporkan kesalahan ke Lambda. Lambda gagal dalam pemanggilan dalam penerbangan dan merobohkan lingkungan eksekusi.

Jalur – `/runtime/restore/error`

Metode – POST

Header

`Lambda-Runtime-Function-Error-Type`— Jenis kesalahan yang ditemui runtime.
Header ini opsional. Lambda menerima nilai string apa pun; kami sarankan menggunakan format`<Category.Reason>`, di mana Kategori adalah `Runtime` atau `Function` dan Alasan dimulai dengan huruf besar (misalnya,). `Runtime.AfterRestoreError` Nilai yang tidak cocok dengan pola ini dinormalisasi ke `Runtime.Unknown` atau`Function.Unknown`.

Kode respons

- 202 - Diterima. Responsnya adalah`{"status":"OK"}`. Runtime harus keluar dari proses.
- 403 — Dilarang. Runtime tidak dalam keadaan yang memungkinkan `/restore/error` (misalnya, `/restore/next` belum dipanggil).
- 404 - tidak SnapStart diaktifkan untuk fungsi ini.
- 500 – Kesalahan penampung. Lingkungan eksekusi berada dalam keadaan tidak dapat dipulihkan. Keluar dari proses runtime.

Sintaksis Permintaan

```
POST /2018-06-01/runtime/restore/error HTTP/1.1
Host: ${AWS_LAMBDA_RUNTIME_API}
Lambda-Runtime-Function-Error-Type: Runtime.AfterRestoreError
```

Sintaksis Respons

```
HTTP/1.1 202 Accepted
Content-Type: application/json

{"status":"OK"}
```

Kapan menggunakan runtime Lambda OS-only

Lambda menyediakan runtime terkelola untuk Java, Python,, .NET Node.js, dan Ruby. Untuk membuat Lambda Event Functions dalam bahasa pemrograman yang tidak tersedia sebagai runtime terkelola, gunakan runtime (OS-only keluarga runtime). `provided` Ada tiga kasus penggunaan utama untuk OS-only runtime:

- Kompilasi native ahead-of-time (AOT): Bahasa seperti Go, Rust, Swift, dan C++ dikompilasi secara native ke biner yang dapat dieksekusi, yang tidak memerlukan runtime bahasa khusus. Bahasa-bahasa ini hanya membutuhkan lingkungan OS di mana biner yang dikompilasi dapat berjalan. Anda juga dapat menggunakan OS-only runtime Lambda untuk menyebarkan binari yang dikompilasi dengan.NET Native AOT dan Java GraalVM Native Image.

  Anda harus menyertakan klien antarmuka runtime dalam biner Anda. Klien antarmuka runtime memanggil Menggunakan API runtime Lambda untuk runtime kustom untuk mengambil pemanggilan fungsi dan kemudian memanggil penangan fungsi Anda. Lambda menyediakan klien antarmuka runtime untuk Rust, Go, .NET Native AOT, Swift (eksperimental), dan C ++ (eksperimental).

  Anda harus mengkompilasi biner Anda untuk lingkungan Linux dan untuk arsitektur set instruksi yang sama yang Anda rencanakan untuk digunakan untuk fungsi (x86_64 atau arm64).
- Third-party runtime: Anda dapat menjalankan fungsi menggunakan runtime off-the-shelf seperti Bref untuk PHP.
- Runtime kustom: Anda dapat membuat runtime sendiri untuk versi bahasa atau bahasa yang Lambda tidak menyediakan runtime terkelola, seperti 19. Node.js Untuk informasi selengkapnya, lihat Membangun runtime khusus untuk AWS Lambda Fungsi Acara. Ini adalah kasus penggunaan yang paling tidak umum untuk OS-only runtime.

Lambda mendukung runtime berikut: OS-only

Nama	Pengidentifikasi	Sistem operasi	Tanggal pengusangan	Buat fungsi blok	Pembaruan fungsi blok
OS-only Runtime	`provided.al2023`	Amazon Linux 2023	30 Jun 2029	Jul 31, 2029	Agustus 31, 2029

Runtime Amazon Linux 2023 (`provided.al2023`) memberikan beberapa keunggulan dibandingkan Amazon Linux 2, termasuk jejak penyebaran yang lebih kecil dan versi pustaka yang diperbarui seperti. `glibc`

`provided.al2023`Runtime digunakan `dnf` sebagai manajer paket, bukan`yum`, yang merupakan manajer paket default di Amazon Linux 2. Untuk informasi selengkapnya tentang perbedaan antara `provided.al2023` dan`provided.al2`, lihat Memperkenalkan runtime Amazon Linux 2023 untuk AWS Lambda di Blog AWS Komputasi.

Membangun runtime khusus untuk AWS Lambda Fungsi Acara

Anda dapat menerapkan AWS Lambda runtime dalam bahasa pemrograman apa pun. Runtime adalah program yang menjalankan metode handler Lambda Event Function saat fungsi dipanggil. Anda dapat menyertakan runtime dalam paket penerapan fungsi Anda atau mendistribusikannya dalam lapisan. Saat Anda membuat fungsi, pilih OS-only runtime (keluarga `provided` runtime).

> **Note**
>
> Membuat runtime khusus adalah kasus penggunaan lanjutan. Jika Anda mencari informasi tentang kompilasi ke biner asli atau menggunakan runtime pihak ketiga yang tersedia, lihat. Kapan menggunakan runtime Lambda OS-only

Untuk panduan proses penerapan runtime kustom, lihat. Tutorial: Membangun runtime khusus

Topik

- Persyaratan
- Menerapkan streaming respons dalam runtime khusus
- Membangun runtime kustom untuk Instans Terkelola Lambda

Persyaratan

Runtime khusus harus menyelesaikan tugas inisialisasi dan pemrosesan tertentu. Runtime menjalankan kode penyiapan fungsi, membaca nama handler dari variabel lingkungan, dan membaca peristiwa pemanggilan dari API runtime Lambda. Runtime meneruskan data event ke handler fungsi, dan memposting respons dari handler kembali ke Lambda.

Tugas inisialisasi

Tugas inisialisasi dijalankan sekali per instans fungsi untuk menyiapkan lingkungan untuk menangani invokasi.

- Ambil pengaturan – Baca variabel lingkungan untuk mendapatkan detail tentang fungsi dan lingkungan.
  - `_HANDLER` – Lokasi ke handler, dari konfigurasi fungsi. Format standar adalah *`file`*`.`*`method`*, dengan `file` adalah nama file tanpa ekstensi, dan `method` merupakan nama metode atau fungsi yang ditentukan dalam file.
  - `LAMBDA_TASK_ROOT` – Direktori yang berisi kode fungsi.
  - `AWS_LAMBDA_RUNTIME_API` – Host dan port API runtime.

  Untuk daftar lengkap variabel yang tersedia, lihatVariabel lingkungan runtime yang ditetapkan.
- Inisialisasi fungsi – Muat file handler dan jalankan kode global atau statis yang ada di dalamnya. Fungsi harus membuat sumber daya statis seperti klien SDK dan koneksi database satu kali, dan menggunakannya kembali untuk beberapa invokasi.
- Mengatasi kesalahan – Jika terjadi kesalahan, hubungi kesalahan inisialisasi API dan segera keluar.

Inisialisasi diperhitungkan dalam waktu dan batas waktu eksekusi yang ditagih. Saat eksekusi memicu inisialisasi instans baru dari fungsi, Anda dapat melihat waktu inisialisasi dalam log dan jejak AWS X-Ray.

Example log

```
REPORT RequestId: f8ac1208... Init Duration: 48.26 ms   Duration: 237.17 ms   Billed
 Duration: 300 ms   Memory Size: 128 MB   Max Memory Used: 26 MB
```

Memproses tugas

Saat belajar, runtime menggunakan antarmuka runtime Lambda untuk mengelola event masuk dan melaporkan kesalahan. Setelah menyelesaikan tugas inisialisasi, runtime memproses event masuk secara berulang. Dalam kode runtime Anda, lakukan langkah berikut secara berurutan.

- Dapatkan event – Hubungi API invokasi beikutnya untuk mendapatkan event berikutnya. Badan respons berisi data event. Header respons berisi ID permintaan dan informasi lainnya.

- Menyebarkan header tracing — Dapatkan header X-Ray tracing dari `Lambda-Runtime-Trace-Id` header dalam respons API. Tetapkan variabel lingkungan `_X_AMZN_TRACE_ID` secara lokal dengan nilai yang sama. X-Ray SDK menggunakan nilai ini untuk menghubungkan data jejak antar layanan.
- Buat objek konteks – Buat objek dengan informasi konteks dari variabel lingkungan dan header dalam respons API.
- Memanggil handler fungsi – Berikan event dan objek konteks ke handler.
- Menangani respons – Hubungi respons pemanggil API untuk memposting respons dari penanggung jawab.
- Mengatasi kesalahan – Jika terjadi kesalahan, hubungi kesalahan invokasi API.
- Pembersihan – Lepaskan sumber daya yang tidak digunakan, kirim data ke layanan lain, atau lakukan tugas tambahan sebelum melanjutkan ke event berikutnya.

Entrypoint

Titik entri runtime kustom adalah file eksekusi bernama `bootstrap`. File bootstrap dapat menjadi runtime, atau dapat memanggil file lain yang membuat runtime. Jika root paket penyebaran Anda tidak berisi file bernama`bootstrap`, Lambda mencari file di lapisan fungsi. Jika `bootstrap` file tidak ada atau tidak dapat dieksekusi, fungsi Anda mengembalikan `Runtime.InvalidEntrypoint` kesalahan saat pemanggilan.

Berikut adalah contoh `bootstrap` file yang menggunakan versi bundel Node.js untuk menjalankan JavaScript runtime dalam file terpisah bernama. `runtime.js`

Example bootstrap

```
#!/bin/sh
    cd $LAMBDA_TASK_ROOT
    ./node-v11.1.0-linux-x64/bin/node runtime.js
```

Menerapkan streaming respons dalam runtime khusus

Untuk fungsi streaming respons, titik akhir `response` dan `error` titik akhir memiliki perilaku yang sedikit dimodifikasi yang memungkinkan runtime mengalirkan sebagian respons ke klien dan mengembalikan muatan dalam potongan. Untuk informasi selengkapnya tentang perilaku tertentu, lihat berikut ini:

- `/runtime/invocation/AwsRequestId/response`— Menyebarkan `Content-Type` header dari runtime untuk dikirim ke klien. Lambda mengembalikan payload respons dalam potongan dengan menggunakan pengkodean transfer chunked. HTTP/1.1 Untuk melakukan streaming respons ke Lambda, runtime harus:
  - Atur header `Lambda-Runtime-Function-Response-Mode` HTTP `kestreaming`.
  - Atur header `Transfer-Encoding` ke `chunked`.
  - Tulis respons yang sesuai dengan spesifikasi pengkodean transfer HTTP/1.1 chunked.
  - Tutup koneksi yang mendasarinya setelah berhasil menulis respons.
- `/runtime/invocation/AwsRequestId/error`— Runtime dapat menggunakan titik akhir ini untuk melaporkan kesalahan fungsi atau runtime ke Lambda, yang juga menerima header. `Transfer-Encoding` Titik akhir ini hanya dapat dipanggil sebelum runtime mulai mengirimkan respons pemanggilan.
- Laporkan kesalahan midstream menggunakan trailer kesalahan di `/runtime/invocation/AwsRequestId/response` — Untuk melaporkan kesalahan yang terjadi setelah runtime mulai menulis respons pemanggilan, runtime dapat secara opsional melampirkan header trailing HTTP bernama dan. `Lambda-Runtime-Function-Error-Type` `Lambda-Runtime-Function-Error-Body` Lambda memperlakukan ini sebagai respons yang berhasil dan meneruskan metadata kesalahan yang disediakan runtime kepada klien.

  > **Note**
  >
  > Untuk melampirkan header tambahan, runtime harus menetapkan nilai `Trailer` header di awal permintaan HTTP. Ini adalah persyaratan dari spesifikasi pengkodean transfer HTTP/1.1 chunked.

  - `Lambda-Runtime-Function-Error-Type`— Jenis kesalahan yang ditemui runtime. Header ini terdiri dari nilai string. Lambda menerima string apa pun, tetapi kami merekomendasikan format. *`<category.reason>`* Misalnya, `Runtime.APIKeyNotFound`.
  - `Lambda-Runtime-Function-Error-Body`— Base64-encoded informasi tentang kesalahan.

Membangun runtime kustom untuk Instans Terkelola Lambda

Instans Terkelola Lambda menggunakan API runtime yang sama dengan fungsi yang menggunakan penyedia kapasitas default Lambda. Namun, ada perbedaan utama dalam cara runtime kustom harus diimplementasikan untuk mendukung model eksekusi bersamaan dari Instans Terkelola.

Penanganan permintaan bersamaan

Perbedaan utama saat membuat runtime kustom untuk Instans Terkelola adalah dukungan untuk pemanggilan bersamaan. Tidak seperti fungsi yang menggunakan penyedia kapasitas default Lambda, di mana runtime memproses satu pemanggilan pada satu waktu, Instans Terkelola dapat memproses beberapa pemanggilan secara bersamaan dalam satu lingkungan eksekusi.

Runtime kustom Anda harus:

- Mendukung **`/next`** permintaan bersamaan - Runtime dapat membuat beberapa panggilan simultan ke API pemanggilan berikutnya, hingga batas yang ditentukan oleh variabel lingkungan. `AWS_LAMBDA_MAX_CONCURRENCY`
- Menangani **`/response`** permintaan bersamaan - Beberapa pemanggilan dapat memanggil API respons pemanggilan secara bersamaan.
- Menerapkan penanganan permintaan aman utas - Pastikan pemanggilan bersamaan tidak saling mengganggu dengan mengelola sumber daya dan status bersama dengan benar.
- Gunakan ID permintaan dan ID pemanggilan dengan benar — ID permintaan mengidentifikasi peristiwa; gunakan di jalur URL. ID pemanggilan mewakili satu upaya pemanggilan—satu ID permintaan dapat menghasilkan beberapa upaya, masing-masing dengan ID pemanggilan unik. Gema kembali ID pemanggilan dan panggilan. `/response` `/error` Header bersifat opsional—menghilangkannya diterima untuk kompatibilitas mundur. Lambda hanya menolak dengan `400` `InvalidInvocationId` ketika header hadir tetapi nilainya tidak cocok.

Pola implementasi

Pola implementasi khas untuk runtime Instans Terkelola melibatkan pembuatan thread atau proses pekerja untuk menangani pemanggilan bersamaan:

1. Baca batas konkurensi — Pada inisialisasi, baca variabel `AWS_LAMBDA_MAX_CONCURRENCY` lingkungan untuk menentukan berapa banyak pemanggilan bersamaan yang akan didukung.
2. Buat kumpulan pekerja — Inisialisasi kumpulan pekerja (utas, proses, atau tugas asinkron) sama dengan batas konkurensi.

3. Loop pemrosesan pekerja - Setiap pekerja secara independen:
   - Panggilan `/runtime/invocation/next` untuk mendapatkan acara pemanggilan
   - Memanggil fungsi handler dengan data peristiwa
   - Memposting tanggapan terhadap `/runtime/invocation/AwsRequestId/response`
   - Mengulangi loop

Pertimbangan tambahan

- Format logging - Instans Terkelola hanya mendukung format log JSON. Pastikan runtime Anda menghormati variabel `AWS_LAMBDA_LOG_FORMAT` lingkungan dan hanya menggunakan format JSON. Untuk informasi selengkapnya, lihat Mengkonfigurasi JSON dan format log teks biasa.
- Sumber daya bersama — Berhati-hatilah saat menggunakan sumber daya bersama seperti `/tmp` direktori dengan pemanggilan bersamaan. Menerapkan mekanisme penguncian yang tepat untuk mencegah kondisi balapan.

Untuk informasi selengkapnya tentang lingkungan eksekusi Instans Terkelola Lambda, lihat. Memahami lingkungan eksekusi Instans Terkelola Lambda

Tutorial: Membangun runtime khusus

Dalam tutorial ini, Anda membuat Lambda Event Function dengan runtime kustom. Anda memulai dengan memasukkan waktu pengoperasian dalam paket deployment fungsi. Kemudian Anda memigrasikannya ke lapisan yang Anda kelola secara terpisah dari fungsi. Terakhir, Anda membagikan lapisan waktu pengoperasian kepada dunia dengan memperbarui kebijakan izin berbasis sumber daya.

Prasyarat

Tutorial ini mengasumsikan bahwa Anda memiliki pengetahuan tentang operasi Lambda dan konsol Lambda dasar. Jika belum, ikuti petunjuk di Membuat fungsi Lambda dengan konsol untuk membuat fungsi Lambda pertama Anda.

Untuk menyelesaikan langkah-langkah berikut, Anda memerlukan AWS CLI versi 2. Perintah dan output yang diharapkan dicantumkan dalam blok terpisah:

```
aws --version
```

Anda akan melihat output berikut:

```
aws-cli/2.13.27 Python/3.11.6 Linux/4.14.328-248.540.amzn2.x86_64 exe/x86_64.amzn.2
```

Untuk perintah panjang, karakter escape (\) digunakan untuk memisahkan perintah menjadi beberapa baris.

Di Linux dan macOS, gunakan shell dan manajer paket pilihan Anda.

> **Note**
>
> Di Windows, beberapa perintah Bash CLI yang biasa Anda gunakan dengan Lambda (`zip`seperti) tidak didukung oleh terminal bawaan sistem operasi. Untuk mendapatkan Windows-integrated versi Ubuntu dan Bash, instal Subsistem Windows untuk Linux. Contoh perintah CLI dalam panduan ini menggunakan pemformatan Linux. Perintah yang menyertakan dokumen JSON sebaris harus diformat ulang jika Anda menggunakan CLI Windows.

Anda memerlukan IAM role untuk membuat fungsi Lambda. Peran memerlukan izin untuk mengirim CloudWatch log ke Log dan mengakses Layanan AWS yang digunakan fungsi Anda. Jika Anda tidak memiliki peran untuk pengembangan fungsi, buatlah sekarang juga.

Untuk membuat peran eksekusi

1. Buka halaman peran di konsol IAM.
2. Pilih Buat peran.
3. Buat peran dengan properti berikut.

   - Entitas tepercaya – Lambda.
   - Izin – AWSLambdaBasicExecutionRole.
   - Nama peran – **`lambda-role`**.

   AWSLambdaBasicExecutionRoleKebijakan ini memiliki izin yang diperlukan fungsi untuk menulis log ke CloudWatch Log.

Membuat fungsi

Buat Fungsi Acara Lambda dengan runtime khusus. Contoh ini mencakup dua file: file runtime `bootstrap` dan function handler. Keduanya diterapkan di Bash.

1. Buat direktori untuk proyek, dan kemudian beralih ke direktori itu.

```
mkdir runtime-tutorial
cd runtime-tutorial
```

2. Buat file baru bernama`bootstrap`. Ini adalah runtime khusus.

   Example bootstrap

```
#!/bin/sh

set -euo pipefail

# Initialization - load function handler
source $LAMBDA_TASK_ROOT/"$(echo $_HANDLER | cut -d. -f1).sh"

# Processing
while true
do
  HEADERS="$(mktemp)"
  # Get an event. The HTTP request will block until one is received
  EVENT_DATA=$(curl -sS -LD "$HEADERS" "http://
${AWS_LAMBDA_RUNTIME_API}/2018-06-01/runtime/invocation/next")

  # Extract request ID by scraping response headers received above
  REQUEST_ID=$(grep -Fi Lambda-Runtime-Aws-Request-Id "$HEADERS" | tr -d
 '[:space:]' | cut -d: -f2)

  # Extract invocation ID for cross-wiring protection
  INVOCATION_ID=$(grep -Fi Lambda-Runtime-Invocation-Id "$HEADERS" | tr -d
 '[:space:]' | cut -d: -f2)

  # Run the handler function from the script
  RESPONSE=$($(echo "$_HANDLER" | cut -d. -f2) "$EVENT_DATA")

  # Send the response, echoing the invocation ID
  curl "http://${AWS_LAMBDA_RUNTIME_API}/2018-06-01/runtime/invocation/$REQUEST_ID/
response"  -d "$RESPONSE" \
```

```
    ${INVOCATION_ID:+--header "Lambda-Runtime-Invocation-Id: $INVOCATION_ID"}
done
```

Waktu pengoperasian memuat skrip fungsi dari paket deployment. Runtime menggunakan dua variabel untuk menemukan skrip.`LAMBDA_TASK_ROOT` memberi tahu tempat paket diekstrak, dan `_HANDLER` mencakup nama skrip.

Setelah runtime memuat skrip fungsi, ia menggunakan API runtime untuk mengambil peristiwa pemanggilan dari Lambda, meneruskan acara ke handler, dan memposting respons kembali ke Lambda. Untuk mendapatkan ID permintaan, waktu pengoperasian menyimpan header dari respons API ke file sementara, dan membaca header `Lambda-Runtime-Aws-Request-Id` dari file. Itu juga membaca `Lambda-Runtime-Invocation-Id` header dan menggemakannya kembali pada respons.

> **Note**
>
> Waktu pengoperasian memiliki tanggung jawab tambahan, termasuk penanganan kesalahan, dan memberikan informasi konteks kepada penangan. Untuk detailnya, lihat Persyaratan.

3. Buat skrip untuk fungsi tersebut. Contoh skrip berikut mendefinisikan fungsi handler yang mengambil data peristiwa, log `kestderr`, dan mengembalikannya.

   Example function.sh

   ```
   function handler () {
     EVENT_DATA=$1
     echo "$EVENT_DATA" 1>&2;
     RESPONSE="Echoing request: '$EVENT_DATA'"

     echo $RESPONSE
   }
   ```

   `runtime-tutorial`Direktori sekarang akan terlihat seperti ini:

   ```
   runtime-tutorial
   # bootstrap
   # function.sh
   ```

4. Buat file dapat dijalankan dan tambahkan ke arsip file .zip. Ini adalah paket penyebaran.

```
chmod 755 function.sh bootstrap
zip function.zip function.sh bootstrap
```

5. Buat fungsi bernama `bash-runtime`. Untuk `--role`, masukkan ARN peran eksekusi Lambda Anda.

```
aws lambda create-function --function-name bash-runtime \
--zip-file fileb://function.zip --handler function.handler --runtime
 provided.al2023 \
--role arn:aws:iam::123456789012:role/lambda-role
```

6. Memanggil fungsi.

```
aws lambda invoke --function-name bash-runtime --payload '{"text":"Hello"}'
 response.txt --cli-binary-format raw-in-base64-out
```

cli-binary-formatOpsi ini diperlukan jika Anda menggunakan AWS CLI versi 2. Untuk menjadikan ini pengaturan default, jalankan`aws configure set cli-binary-format raw-in-base64-out`. Untuk informasi selengkapnya, lihat opsi baris perintah global yang AWS CLI didukung di Panduan AWS Command Line Interface Pengguna untuk Versi 2.

Anda akan melihat tanggapan seperti ini:

```
{
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}
```

7. Verifikasi responsnya.

```
cat response.txt
```

Anda akan melihat tanggapan seperti ini:

```
Echoing request: '{"text":"Hello"}'
```

Buat lapisan

Untuk memisahkan kode waktu pengoperasian dari kode fungsi, buat lapisan yang hanya berisi waktu pengoperasian. Lapisan memungkinkan Anda mengembangkan dependensi fungsi secara independen, dan dapat mengurangi penggunaan penyimpanan ketika Anda menggunakan lapisan yang sama dengan beberapa fungsi. Untuk informasi selengkapnya, lihat Mengelola dependensi Fungsi Acara Lambda dengan lapisan.

1. Buat file.zip yang berisi `bootstrap` file.

   ```
   zip runtime.zip bootstrap
   ```

2. Buat layer dengan perintah publish-layer-version.

   ```
   aws lambda publish-layer-version --layer-name bash-runtime --zip-file fileb://
   runtime.zip
   ```

   Ini menciptakan versi pertama lapisan.

Perbarui fungsi

Untuk menggunakan layer runtime dalam fungsi, konfigurasikan fungsi untuk menggunakan layer, dan hapus kode runtime dari fungsi.

1. Perbarui konfigurasi fungsi untuk menarik lapisan.

   ```
   aws lambda update-function-configuration --function-name bash-runtime \
   --layers arn:aws:lambda:us-east-1:123456789012:layer:bash-runtime:1
   ```

   Ini menambahkan waktu pengoperasian ke fungsi di direktori `/opt`. Untuk memastikan bahwa Lambda menggunakan runtime di layer, Anda harus menghapus `boostrap` dari paket penerapan fungsi, seperti yang ditunjukkan dalam dua langkah berikutnya.

2. Buat file.zip yang berisi kode fungsi.

   ```
   zip function-only.zip function.sh
   ```

3. Perbarui kode fungsi untuk hanya memasukkan skrip penangan.

```
aws lambda update-function-code --function-name bash-runtime --zip-file fileb://
function-only.zip
```

4. Panggil fungsi untuk mengonfirmasi bahwa ia berfungsi dengan layer runtime.

```
aws lambda invoke --function-name bash-runtime --payload '{"text":"Hello"}'
 response.txt --cli-binary-format raw-in-base64-out
```

cli-binary-formatOpsi ini diperlukan jika Anda menggunakan AWS CLI versi 2. Untuk menjadikan ini pengaturan default, jalankan`aws configure set cli-binary-format raw-in-base64-out`. Untuk informasi selengkapnya, lihat opsi baris perintah global yang AWS CLI didukung di Panduan AWS Command Line Interface Pengguna untuk Versi 2.

Anda akan melihat tanggapan seperti ini:

```
{
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}
```

5. Verifikasi responsnya.

```
cat response.txt
```

Anda akan melihat tanggapan seperti ini:

```
Echoing request: '{"text":"Hello"}'
```

Perbarui waktu pengoperasian

1. Untuk mencatat log informasi tentang lingkungan eksekusi, perbarui skrip waktu pengoperasian ke variabel lingkungan output.

Example bootstrap

```
#!/bin/sh

set -euo pipefail
```

```
# Configure runtime to output environment variables
echo "##  Environment variables:"
env

# Load function handler
source $LAMBDA_TASK_ROOT/"$(echo $_HANDLER | cut -d. -f1).sh"

# Processing
while true
do
  HEADERS="$(mktemp)"
  # Get an event. The HTTP request will block until one is received
  EVENT_DATA=$(curl -sS -LD "$HEADERS" "http://
${AWS_LAMBDA_RUNTIME_API}/2018-06-01/runtime/invocation/next")

  # Extract request ID by scraping response headers received above
  REQUEST_ID=$(grep -Fi Lambda-Runtime-Aws-Request-Id "$HEADERS" | tr -d
 '[:space:]' | cut -d: -f2)

  # Extract invocation ID for cross-wiring protection
  INVOCATION_ID=$(grep -Fi Lambda-Runtime-Invocation-Id "$HEADERS" | tr -d
 '[:space:]' | cut -d: -f2)

  # Run the handler function from the script
  RESPONSE=$($(echo "$_HANDLER" | cut -d. -f2) "$EVENT_DATA")

  # Send the response, echoing the invocation ID
  curl "http://${AWS_LAMBDA_RUNTIME_API}/2018-06-01/runtime/invocation/$REQUEST_ID/
response"  -d "$RESPONSE" \
    ${INVOCATION_ID:+--header "Lambda-Runtime-Invocation-Id: $INVOCATION_ID"}
done
```

2. Buat file.zip yang berisi versi baru `bootstrap` file.

```
zip runtime.zip bootstrap
```

3. Buat versi baru dari `bash-runtime` layer.

```
aws lambda publish-layer-version --layer-name bash-runtime --zip-file fileb://
runtime.zip
```

4. Konfigurasikan fungsi untuk menggunakan versi baru lapisan.

```
aws lambda update-function-configuration --function-name bash-runtime \
--layers arn:aws:lambda:us-east-1:123456789012:layer:bash-runtime:2
```

Bagikan lapisan

Untuk berbagi layer dengan yang lain Akun AWS, tambahkan pernyataan izin lintas akun ke kebijakan berbasis sumber daya lapisan. Jalankan perintah add-layer-version-permission dan tentukan ID akun sebagai. `principal` Dalam setiap pernyataan, Anda dapat memberikan izin ke satu akun, semua akun, atau organisasi di AWS Organizations.

Contoh berikut memberikan akun 111122223333 akses ke versi 2 lapisan. `bash-runtime`

```
aws lambda add-layer-version-permission \
  --layer-name bash-runtime \
  --version-number 2 \
  --statement-id xaccount \
  --action lambda:GetLayerVersion \
  --principal 111122223333 \
  --output text
```

Anda akan melihat output yang serupa dengan yang berikut:

```
{"Sid":"xaccount","Effect":"Allow","Principal":
{"AWS":"arn:aws:iam::111122223333:root"},"Action":"lambda:GetLayerVersion","Resource":"arn:aws:
east-1:123456789012:layer:bash-runtime:2"}
```

Izin hanya berlaku untuk satu versi lapisan. Ulangi proses ini tiap kali Anda membuat versi lapisan baru.

Bersihkan

Hapus setiap versi lapisan.

```
aws lambda delete-layer-version --layer-name bash-runtime --version-number 1
aws lambda delete-layer-version --layer-name bash-runtime --version-number 2
```

Karena fungsi memiliki referensi ke lapisan versi 2, ia masih ada di Lambda. Fungsi terus bekerja, tetapi fungsi tidak dapat lagi dikonfigurasi untuk menggunakan versi yang dihapus. Jika Anda

memodifikasi daftar lapisan pada fungsi, Anda harus menentukan versi baru atau menghilangkan lapisan yang dihapus.

Hapus fungsi dengan perintah hapus-fungsi.

```
aws lambda delete-function --function-name bash-runtime
```

Repositori sumber terbuka

AWS Lambda menyediakan berbagai alat open source, pustaka, dan komponen untuk membantu Anda membangun, menyesuaikan, dan mengoptimalkan aplikasi tanpa server Anda. Sumber daya ini termasuk klien antarmuka runtime, pustaka acara, gambar dasar kontainer, alat pengembangan, dan proyek sampel yang dikelola oleh AWS dan tersedia di. GitHub Dengan menggunakan repositori open source ini, Anda dapat memperluas kemampuan Lambda, membuat runtime kustom, memproses peristiwa dari berbagai AWS layanan, dan mendapatkan wawasan yang lebih dalam tentang kinerja fungsi Anda. Halaman ini memberikan ikhtisar proyek open source utama yang mendukung pengembangan Lambda.

Klien Antarmuka Runtime

Lambda Runtime Interface Clients (RICs) adalah pustaka open source yang mengimplementasikan Runtime API dan mengelola interaksi antara kode fungsi Anda dan layanan Lambda. Klien ini menangani menerima acara pemanggilan, meneruskan informasi konteks, dan melaporkan kesalahan.

Klien antarmuka runtime yang digunakan oleh runtime terkelola Lambda dan gambar dasar kontainer diterbitkan sebagai open source. Saat Anda membuat runtime kustom atau memperpanjang runtime yang sudah ada, Anda dapat menggunakan pustaka open source ini untuk menyederhanakan implementasi Anda. GitHub Repositori open source berikut berisi kode sumber untuk RIC Lambda:

- Node.js Klien Antarmuka Runtime
- Klien Antarmuka Runtime Python
- Klien Antarmuka Java Runtime
- Klien Antarmuka Ruby Runtime
- Klien Antarmuka Runtime .NET
- Klien Antarmuka Rust Runtime

- Klien Antarmuka Go Runtime
- Klien Antarmuka Runtime Swift (eksperimental)
- Klien Antarmuka Runtime C ++ (eksperimental)
- Gambar Lambda Base

Untuk informasi selengkapnya tentang menggunakan klien ini untuk membangun runtime kustom, lihatthe section called “Membangun runtime kustom”.

Perpustakaan acara

Pustaka acara Lambda menyediakan definisi tipe dan utilitas pembantu untuk memproses acara dari berbagai layanan. AWS Pustaka ini membantu Anda mengurai dan menangani data peristiwa dengan cara yang aman untuk jenis, membuatnya lebih mudah untuk bekerja dengan peristiwa dari layanan seperti Amazon S3, Amazon DynamoDB, dan Amazon API Gateway.

Untuk bahasa yang dikompilasi, AWS sediakan pustaka acara berikut:

- Perpustakaan Acara Java
- .NET Perpustakaan Acara
- Perpustakaan Acara Go
- Perpustakaan Acara Rust

Untuk bahasa yang ditafsirkan seperti Node.js, Python, dan Ruby, peristiwa dapat diuraikan secara langsung sebagai objek JSON tanpa memerlukan perpustakaan terpisah. Namun, pengembang yang menggunakan Node.js dan Python dapat menggunakan powertools untuk AWS Lambda, yang menyediakan skema bawaan untuk AWS acara yang menawarkan petunjuk tipe, validasi data, dan fungsionalitas yang mirip dengan apa yang disediakan perpustakaan bahasa yang dikompilasi.

- Powertools untuk TypeScript
- Powertools untuk Python

Gambar dasar kontainer

AWS menyediakan gambar dasar wadah open source yang dapat Anda gunakan sebagai titik awal untuk membangun gambar kontainer untuk Fungsi Acara Lambda Anda. Gambar dasar ini mencakup

klien antarmuka runtime dan komponen lain yang diperlukan untuk menjalankan fungsi Anda di lingkungan eksekusi Lambda.

Untuk informasi selengkapnya tentang gambar dasar yang tersedia dan cara menggunakannya, lihat repositori Gambar Dasar AWS Lambda dan. the section called “Image kontainer”

Alat pengembangan

AWS menyediakan alat pengembangan open source tambahan untuk membantu Anda membangun dan mengoptimalkan fungsi Lambda Anda:

Powertools untuk AWS Lambda

Powertools untuk AWS Lambda menyederhanakan pengembangan tanpa server dengan utilitas penting untuk mencegah pemrosesan duplikat, dan pemrosesan batch untuk penanganan multi-rekaman dan perpustakaan konsumen Kafka. Fitur-fitur ini membantu Anda meminimalkan kompleksitas kode dan overhead operasional.

Anda juga dapat menggunakan validasi skema acara bawaan, pencatatan dan penelusuran terstruktur, dan integrasi penyimpanan parameter yang dirancang untuk mempercepat pembuatan fungsi Lambda siap produksi sambil mengikuti praktik terbaik yang dirancang dengan baik. AWS

GitHub repositori:

- Python
- TypeScript
- Java
- .NET

Alat pengembangan Java

- Java Profiler (eksperimental) - Alat untuk membuat profil fungsi Java Lambda.
- Java Libraries - Sebuah repositori yang berisi koleksi komprehensif perpustakaan Java dan alat-alat untuk pengembangan Lambda, termasuk proyek-proyek utama seperti utilitas pengujian JUnit dan alat profiling.
- Serverless Java Container - Pustaka yang dapat Anda gunakan untuk menjalankan aplikasi Java yang ada di Lambda dengan perubahan minimal.

Alat pengembangan.NET

AWS Lambda Repositori.NET menyediakan pustaka dan alat .NET untuk pengembangan Lambda, termasuk proyek-proyek utama seperti untuk alat AWS Lambda untuk server.NET CLI dan .NET Core untuk hosting aplikasi.NET Core.

Contoh proyek

Jelajahi koleksi komprehensif proyek dan aplikasi Lambda sampel di repositori Serverless Land. Sampel ini menunjukkan berbagai kasus penggunaan Lambda, pola integrasi, dan praktik terbaik untuk membantu Anda memulai aplikasi tanpa server.

Melakukan konfigurasi AWS Lambda Fungsi Acara

Pelajari cara mengonfigurasi kemampuan dan opsi inti untuk Fungsi Peristiwa Lambda Anda menggunakan API atau konsol Lambda.

arsip file.zip

Buat paket penerapan fungsi Lambda saat Anda ingin menyertakan dependensi, lapisan runtime kustom, atau file apa pun di luar kode fungsi Anda. Paket deployment adalah arsip file.zip yang berisi kode fungsi dan dependensi Anda.

Gambar kontainer

Gunakan gambar kontainer untuk mengemas kode fungsi dan dependensi saat Anda memerlukan kontrol lebih besar atas proses pembuatan, atau jika fungsi Anda memerlukan konfigurasi runtime khusus. Anda dapat membangun, menguji, dan menerapkan fungsi Lambda sebagai gambar kontainer menggunakan alat seperti Docker CLI.

Self-managed Penyimpanan kode S3

Konfigurasikan fungsi dan lapisan Anda untuk menggunakan bucket S3 Anda sendiri untuk penyimpanan kode. Lambda mereferensikan kode Anda langsung dari bucket S3 Anda, menghilangkan batas Lambda-managed penyimpanan dan mengurangi waktu untuk pemanggilan pertama.

Memori

Pelajari bagaimana dan kapan harus meningkatkan memori fungsi.

Penyimpanan fana

Pelajari bagaimana dan kapan harus meningkatkan kapasitas penyimpanan sementara fungsi Anda.

Batas waktu

Pelajari bagaimana dan kapan harus meningkatkan nilai batas waktu fungsi Anda.

Variabel lingkungan

Anda dapat membuat kode fungsi Anda portabel dan menyimpan rahasia dari kode Anda dengan menyimpannya dalam konfigurasi fungsi Anda dengan menggunakan variabel lingkungan.

Jaringan keluar

Anda dapat menggunakan fungsi Lambda Anda dengan AWS sumber daya di VPC Amazon. Menghubungkan fungsi Anda ke VPC memberi Anda akses ke sumber daya di subnet pribadi seperti database relasional dan cache.

Jaringan masuk

Anda dapat menggunakan titik akhir VPC antarmuka untuk menjalankan fungsi Lambda Anda tanpa melintasi internet publik.

Sistem berkas

Anda dapat menggunakan fungsi Lambda untuk memasang Amazon EFS ke direktori lokal. Sistem file memungkinkan kode fungsi Anda untuk mengakses dan memodifikasi sumber daya bersama dengan aman dan pada konkurensi tinggi.

Alias

Anda dapat mengonfigurasi klien Anda untuk memanggil versi fungsi Lambda tertentu dengan menggunakan alias, alih-alih memperbarui klien.

Versi

Dengan menerbitkan versi fungsi Anda, Anda dapat menyimpan kode dan konfigurasi Anda sebagai sumber daya terpisah yang tidak dapat diubah.

Tanda

Gunakan tag untuk mengaktifkan kontrol akses berbasis atribut (ABAC), untuk mengatur fungsi Lambda Anda, dan untuk memfilter dan menghasilkan laporan tentang fungsi Anda menggunakan atau layanan Billing and AWS Cost Explorer Cost AWS Management.

Streaming respons

Anda dapat mengonfigurasi URL fungsi Lambda Anda untuk mengalirkan muatan respons kembali ke klien. Streaming respons dapat menguntungkan aplikasi sensitif latensi dengan meningkatkan kinerja time to first byte (TTFB). Ini karena Anda dapat mengirim sebagian tanggapan kembali ke klien saat tersedia. Selain itu, Anda dapat menggunakan streaming respons untuk membangun fungsi yang mengembalikan muatan yang lebih besar.

Titik akhir metadata

Gunakan titik akhir metadata Lambda untuk menemukan Zona Ketersediaan mana fungsi Anda berjalan, memungkinkan Anda mengoptimalkan latensi dengan merutekan ke sumber daya Same-AZ dan menerapkan pola ketahanan. AZ-aware

Menyebarkan Fungsi Peristiwa Lambda sebagai arsip file.zip

Saat Anda membuat Lambda Event Function, Anda mengemas kode fungsi Anda ke dalam paket penerapan. Lambda mendukung dua tipe paket deployment: gambar kontainer dan arsip file .zip. Alur kerja untuk membuat fungsi tergantung pada jenis paket penerapan. Untuk mengonfigurasi fungsi yang didefinisikan sebagai gambar kontainer, lihat the section called “Image kontainer”.

Anda dapat menggunakan konsol Lambda dan API Lambda untuk membuat fungsi yang ditentukan dengan arsip file.zip. Anda juga dapat mengunggah file.zip yang diperbarui untuk mengubah kode fungsi.

Secara default, saat Anda mengunggah file.zip, Lambda menyimpan salinan kode sumber Anda di Lambda-managed penyimpanan. Dengan penyimpanan kode S3 yang dikelola sendiri, Lambda membaca kode sumber Anda dari bucket S3 Anda selama pembuatan fungsi dan operasi pembaruan, dan tidak menyimpan salinan terpisah dari kode sumber Anda. Lambda masih membuat representasi internal yang dioptimalkan dari kode Anda untuk pemanggilan. Kode sumber Anda di S3 tidak diakses pada setiap start dingin.

> **Note**
>
> Anda tidak dapat mengubah jenis paket penyebaran (.zip atau gambar wadah) untuk fungsi yang ada. Misalnya, Anda tidak dapat mengonversi fungsi gambar kontainer untuk menggunakan arsip berkas.zip. Anda harus membuat fungsi baru.

Topik

- Membuat fungsi
- Menggunakan editor kode konsol
- Memperbarui kode fungsi
- Mengubah runtime
- Mengubah arsitektur
- Menggunakan API Lambda
- Mengunduh kode fungsi Anda
- CloudFormation
- Mengenkripsi paket deployment .zip Lambda

Membuat fungsi

Saat Anda membuat fungsi yang ditentukan dengan arsip file.zip, Anda memilih template kode, versi bahasa, dan peran eksekusi untuk fungsi tersebut. Anda menambahkan kode fungsi Anda setelah Lambda membuat fungsi.

Untuk membuat fungsi

1. Buka Halaman Fungsi Acara dari konsol Lambda.
2. Memilih Buat Fungsi Acara.
3. Memilih Penulis dari awal atau Gunakan cetak biru untuk membuat fungsi Anda.
4. Di bagian Informasi dasar, lakukan hal berikut:
   a. Untuk Nama fungsi, masukkan nama fungsi. Nama fungsi dibatasi hingga 64 karakter panjangnya.
   b. Untuk Waktu pelaksanaan, pilih versi bahasa yang akan digunakan untuk fungsi Anda.
   c. (Opsional) Untuk Arsitektur, pilih arsitektur set instruksi untuk digunakan untuk fungsi Anda. Arsitektur default adalah x86_64. Saat Anda membuat paket penerapan untuk fungsi Anda, pastikan itu kompatibel dengan ini arsitektur set instruksi.
5. (Opsional) Di bagian Izin, luaskan Ubah peran eksekusi default. Anda dapat membuat yang baru Peran eksekusi atau menggunakan peran yang sudah ada.
6. (Opsional) Perluas Pengaturan lanjutan. Anda dapat memilih Konfigurasi penandatanganan kode untuk fungsinya. Anda juga dapat mengonfigurasi (Amazon VPC) untuk fungsi yang akan diakses.
7. Memilih Buat Fungsi Acara.

Lambda menciptakan fungsi baru. Anda sekarang dapat menggunakan konsol untuk menambahkan kode fungsi dan mengkonfigurasi parameter fungsi dan fitur lainnya. Untuk instruksi penerapan kode, lihat halaman handler untuk runtime yang digunakan fungsi Anda.

Node.js

Menyebarkan Fungsi Acara Node.js Lambda dengan arsip file.zip

Python

Bekerja dengan arsip file.zip untuk Fungsi Acara Python Lambda

Ruby

- Menyebarkan Fungsi Acara Ruby Lambda dengan arsip file.zip

Java

- Menyebarkan Fungsi Acara Lambda Java dengan arsip file.zip atau JAR

Go

- Terapkan Fungsi Acara Go Lambda dengan arsip file.zip

C#

- Bangun dan terapkan Fungsi Acara C# Lambda dengan arsip file.zip

PowerShell

- Menyebarkan Fungsi Acara PowerShell Lambda dengan arsip file.zip

Menggunakan editor kode konsol

Konsol membuat fungsi Lambda dengan file sumber tunggal. Untuk bahasa skrip, Anda dapat mengedit file ini dan menambahkan lebih banyak file menggunakan editor kode bawaan. Untuk menyimpan perubahan Anda, pilih Simpan. Selanjutnya, untuk menjalankan kode, pilih Uji.

Saat Anda menyimpan kode fungsi, konsol Lambda membuat paket penyebaran arsip file.zip. Ketika Anda mengembangkan kode fungsi Anda di luar konsol (menggunakan IDE), Anda perlu membuat paket penerapan untuk mengunggah kode Anda ke fungsi Lambda.

Memperbarui kode fungsi

Untuk bahasa scripting (Node.js, Python, dan Ruby), Anda dapat mengedit kode fungsi Anda di editor kode yang disematkan. Jika kode lebih besar dari 3MB, atau jika Anda perlu menambahkan pustaka, atau untuk bahasa yang tidak didukung editor (Java, Go, C#), Anda harus mengunggah kode fungsi Anda sebagai arsip.zip. Jika arsip file.zip berukuran lebih kecil dari 50 MB, Anda dapat mengunggah arsip file.zip dari mesin lokal Anda. Jika file lebih besar dari 50 MB, unggah file ke fungsi dari bucket Amazon S3.

Untuk mengunggah kode fungsi sebagai arsip.zip

1. Buka Halaman Fungsi Acara dari konsol Lambda.

2. Pilih fungsi untuk memperbarui dan pilih Kode tab.
3. Di bagian Sumber kode, pilih Unggah dari.
4. Memilih berkas.zip, dan kemudian memilih Unggah.
   - Di pemilih file, pilih versi gambar baru, pilih Buka, dan kemudian memilih Menyimpan.
5. (Alternatif untuk langkah 4) Pilih Lokasi Amazon S3.
   a. Di kotak teks, masukkan URL tautan S3 dari arsip file.zip.
   b. Di bawah Mode penyimpanan kode, pilih salah satu dari berikut ini:
      - Mode salin (default)
      - Mode referensi
   c. Pilih Simpan.

Saat memperbarui kode fungsi dari bucket Amazon S3 menggunakan AWS CLI, Anda dapat menentukan mode penyimpanan kode. Set `S3ObjectStorageMode` kepada `REFERENCE` untuk menggunakan penyimpanan kode S3 yang dikelola sendiri, atau `COPY` (default) untuk digunakan Lambda-managed penyimpanan. Anda dapat mengubah mode penyimpanan pada setiap pembaruan.

```
aws lambda update-function-code \
  --function-name my-function \
  --s3-bucket my-bucket \
  --s3-key my-function.zip \
  --s3-object-version abc123 \
  --s3-object-storage-mode REFERENCE
```

Mengubah runtime

Jika Anda memperbarui konfigurasi fungsi untuk menggunakan runtime baru, Anda mungkin perlu memperbarui kode fungsi agar kompatibel dengan runtime baru. Jika Anda memperbarui konfigurasi fungsi untuk menggunakan runtime yang berbeda, Anda harus menyediakan kode fungsi baru yang kompatibel dengan runtime dan arsitektur. Untuk petunjuk tentang cara membuat paket penerapan untuk kode fungsi, lihat halaman handler untuk runtime yang digunakan fungsi.

Yang Node.js 20, Python 3.12, Java 21, .NET 8, Ruby 3.3, dan gambar dasar yang lebih baru didasarkan pada image wadah minimal Amazon Linux 2023. Gambar dasar sebelumnya

menggunakan Amazon Linux 2. AL2023 memberikan beberapa keunggulan dibandingkan Amazon Linux 2, termasuk jejak penerapan yang lebih kecil dan versi perpustakaan yang diperbarui seperti `glibc`. Untuk informasi lebih lanjut, lihat Memperkenalkan runtime Amazon Linux 2023 untuk AWS Lambda pada AWS Blog Komputasi.

Untuk mengubah runtime

1. Buka Halaman Fungsi Acara dari konsol Lambda.
2. Pilih fungsi untuk memperbarui dan pilih Kode tab.
3. Gulir ke bawah ke Pengaturan runtime bagian, yang berada di bawah editor kode.
4. Pilih Edit.
    a. Untuk Waktu pelaksanaan, pilih pengidentifikasi runtime.
    b. Untuk Pengendali, tentukan nama file dan handler untuk fungsi Anda.
    c. Untuk Arsitektur, pilih arsitektur set instruksi untuk digunakan untuk fungsi Anda.
5. Pilih Simpan.

Mengubah arsitektur

Sebelum Anda dapat mengubah arsitektur set instruksi, Anda perlu memastikan bahwa kode fungsi Anda kompatibel dengan arsitektur target.

Jika Anda menggunakan Node.js, Python, atau Ruby dan Anda mengedit kode fungsi Anda di editor tertanam, kode yang ada mungkin berjalan tanpa modifikasi.

Namun, jika Anda memberikan kode fungsi menggunakan paket penyebaran arsip berkas.zip, Anda harus menyiapkan arsip berkas.zip baru yang dikompilasi dan dibuat dengan benar untuk runtime target dan arsitektur set instruksi. Untuk instruksi, lihat halaman handler untuk runtime fungsi Anda.

Untuk mengubah arsitektur set instruksi

1. Buka Halaman Fungsi Acara dari konsol Lambda.
2. Pilih fungsi untuk memperbarui dan pilih Kode tab.
3. Di bawah Pengaturan runtime, pilih Sunting.
4. Untuk Arsitektur, pilih arsitektur set instruksi untuk digunakan untuk fungsi Anda.
5. Pilih Simpan.

Menggunakan API Lambda

Untuk membuat dan mengkonfigurasi fungsi yang menggunakan arsip file.zip, gunakan operasi API berikut:

- CreateFunction
- UpdateFunctionCode
- UpdateFunctionConfiguration

Mengunduh kode fungsi Anda

Anda dapat mengunduh yang belum dipublikasikan saat ini (`$LATEST`) versi kode fungsi Anda.zip dari konsol Lambda. Untuk melakukan ini, pertama-tama pastikan Anda memiliki izin IAM berikut:

- `iam:GetPolicy`
- `iam:GetPolicyVersion`
- `iam:GetRole`
- `iam:GetRolePolicy`
- `iam:ListAttachedRolePolicies`
- `iam:ListRolePolicies`
- `iam:ListRoles`

Untuk mengunduh kode fungsi.zip

1. Buka Halaman Fungsi Acara dari konsol Lambda.
2. Pilih fungsi yang ingin Anda unduh kode fungsi.zip untuknya.
3. Di Ikhtisar fungsi, pilih Unduh tombol, lalu pilih Unduh kode fungsi.zip.
    - Atau, pilih Unduh AWS SAM fail untuk menghasilkan dan mengunduh template SAM berdasarkan konfigurasi fungsi Anda. Anda juga dapat memilih Unduh keduanya untuk mengunduh templat.zip dan SAM.

CloudFormation

Anda dapat menggunakan CloudFormation untuk membuat fungsi Lambda yang menggunakan arsip file.zip. Di milikmu CloudFormation template, `AWS::Lambda::Function` resource menentukan

fungsi Lambda. Untuk deskripsi properti di `AWS::Lambda::Function` sumber daya, lihat AWS::Lambda::Fungsi di AWS CloudFormation Panduan Pengguna.

Di `AWS::Lambda::Function` resource, atur properti berikut untuk membuat fungsi yang didefinisikan sebagai arsip file.zip:

- AWS::Lambda::Fungsi
  - PackageType — Setel ke `Zip`.
  - Kode — Masukkan nama bucket Amazon S3 dan nama file.zip di `S3Bucket` dan `S3Key`bidang. Untuk Node.js atau Python, Anda dapat memberikan kode sumber sebaris dari fungsi Lambda Anda.
  - Runtime — Mengatur nilai runtime.
  - Arsitektur — Mengatur nilai arsitektur ke `arm64` untuk menggunakan AWS Prosesor Graviton. Secara default, nilai arsitektur adalah `x86_64`.
  - S3ObjectStorageMode — (Opsional) Setel ke `REFERENCE` untuk menggunakan penyimpanan kode S3 yang dikelola sendiri. Setel ke `COPY` (default) untuk digunakan Lambda-managed penyimpanan.

Mengenkripsi paket deployment .zip Lambda

AWS Lambda selalu menyediakan enkripsi sisi server saat istirahat untuk paket penerapan .zip dan detail konfigurasi fungsi dengan file. AWS KMS key Secara default, Lambda menggunakan file. Kunci milik AWS Jika perilaku default ini sesuai dengan alur kerja Anda, Anda tidak perlu menyiapkan hal lain. AWS tidak membebankan biaya untuk menggunakan kunci ini.

Jika mau, Anda dapat memberikan kunci yang dikelola AWS KMS pelanggan sebagai gantinya. Anda dapat melakukan ini untuk memiliki kontrol atas rotasi kunci KMS atau untuk memenuhi persyaratan organisasi Anda untuk mengelola kunci KMS. Saat Anda menggunakan kunci yang dikelola pelanggan, hanya pengguna di akun Anda dengan akses ke kunci KMS yang dapat melihat atau mengelola kode atau konfigurasi fungsi.

Kunci yang dikelola pelanggan dikenakan AWS KMS biaya standar. Untuk informasi selengkapnya, lihat harga AWS Key Management Service.

Buat kunci terkelola pelanggan

Anda dapat membuat kunci terkelola pelanggan simetris dengan menggunakan Konsol Manajemen AWS, atau AWS KMS API.

Untuk membuat kunci terkelola pelanggan simetris

Ikuti langkah-langkah untuk Membuat enkripsi simetris Membuat kunci KMS simetris di Panduan Pengembang.AWS Key Management Service

Izin

Kebijakan utama

Kebijakan utama mengontrol akses ke kunci yang dikelola pelanggan Anda. Setiap kunci yang dikelola pelanggan harus memiliki persis satu kebijakan utama, yang berisi pernyataan yang menentukan siapa yang dapat menggunakan kunci dan bagaimana mereka dapat menggunakannya. Untuk informasi selengkapnya, lihat Cara mengubah kebijakan utama di Panduan AWS Key Management Service Pengembang.

Saat Anda menggunakan kunci yang dikelola pelanggan untuk mengenkripsi paket penyebaran .zip, Lambda tidak menambahkan hibah ke kunci tersebut. Sebagai gantinya, kebijakan AWS KMS utama Anda harus mengizinkan Lambda memanggil operasi AWS KMS API berikut atas nama Anda:

- km: GenerateDataKey
- kms:Decrypt

Contoh kebijakan kunci berikut memungkinkan semua fungsi Lambda di akun 111122223333 memanggil operasi yang diperlukan AWS KMS untuk kunci terkelola pelanggan yang ditentukan:

Example AWS KMS kebijakan kunci

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "lambda.amazonaws.com"
            },
            "Action": [
                "kms:GenerateDataKey",
                "kms:Decrypt"
            ],
```

```
            "Resource": "arn:aws:kms:us-east-1:111122223333:key/key-id",
            "Condition": {
                "StringLike": {
                "kms:EncryptionContext:aws:lambda:FunctionArn":
 "arn:aws:lambda:us-east-1:111122223333:function:*"
                }
            }
        }
    ]
}
```

Untuk informasi selengkapnya tentang akses kunci pemecahan masalah, lihat Panduan AWS Key Management Service Pengembang.

Izin principal

Saat Anda menggunakan kunci yang dikelola pelanggan untuk mengenkripsi paket penyebaran .zip, hanya prinsipal dengan akses ke kunci tersebut yang dapat mengakses paket penyebaran .zip. Misalnya, prinsipal yang tidak memiliki akses ke kunci yang dikelola pelanggan tidak dapat mengunduh paket.zip menggunakan URL S3 yang telah ditetapkan sebelumnya yang disertakan dalam respons. GetFunction `AccessDeniedException`An dikembalikan di `Code` bagian respons.

Example AWS KMS AccessDeniedException

```
{
    "Code": {
        "RepositoryType": "S3",
        "Error": {
            "ErrorCode": "AccessDeniedException",
            "Message": "KMS access is denied. Check your KMS permissions. KMS
 Exception: AccessDeniedException KMS Message: User: arn:aws:sts::111122223333:assumed-
role/LambdaTestRole/session is not authorized to perform: kms:Decrypt on resource:
 arn:aws:kms:us-east-1:111122223333:key/key-id with an explicit deny in a resource-
based policy"
        },
        "SourceKMSKeyArn": "arn:aws:kms:us-east-1:111122223333:key/key-id"
    },
 ...
```

Untuk informasi selengkapnya tentang izin AWS KMS kunci, lihat Otentikasi dan kontrol akses untuk. AWS KMS

Menggunakan kunci terkelola pelanggan untuk paket penerapan.zip

Gunakan parameter API berikut untuk mengonfigurasi kunci terkelola pelanggan untuk paket penerapan.zip:

- SourceKMSKeyArn: Mengenkripsi paket penyebaran sumber.zip (file yang Anda unggah).
- KMSKeyArn: Mengenkripsi variabel lingkungan dan snapshot Lambda. SnapStart
- DurableConfig.KMSKeyArn: Mengenkripsi data eksekusi yang tahan lama. Kunci ini tidak tergantung pada tingkat fungsi`KMSKeyArn`.

Kapan `SourceKMSKeyArn` dan `KMSKeyArn` keduanya ditentukan, Lambda menggunakan `KMSKeyArn` kunci untuk mengenkripsi versi paket yang tidak di-zip yang digunakan Lambda untuk menjalankan fungsi. Kapan `SourceKMSKeyArn` `KMSKeyArn` ditentukan tetapi tidak, Lambda menggunakan Kunci yang dikelola AWSuntuk mengenkripsi versi paket yang tidak di-zip.

> **Note**
>
> Untuk fungsi dan lapisan yang menggunakan penyimpanan kode S3 yang dikelola sendiri, Lambda tidak menyimpan salinan kode sumber Anda. Kode Anda tetap berada di bucket S3 Anda, dan enkripsi saat istirahat untuk kode sumber Anda dikelola oleh konfigurasi bucket S3 Anda. Lambda akan digunakan `KMSKeyArn` untuk mengenkripsi versi unzip dari paket yang digunakan Lambda untuk menjalankan fungsi. Untuk informasi selengkapnya, lihat Melindungi data dengan enkripsi sisi server di Panduan Pengguna Layanan Penyimpanan Sederhana Amazon.

Lambda console

Untuk menambahkan enkripsi kunci terkelola pelanggan saat Anda membuat fungsi

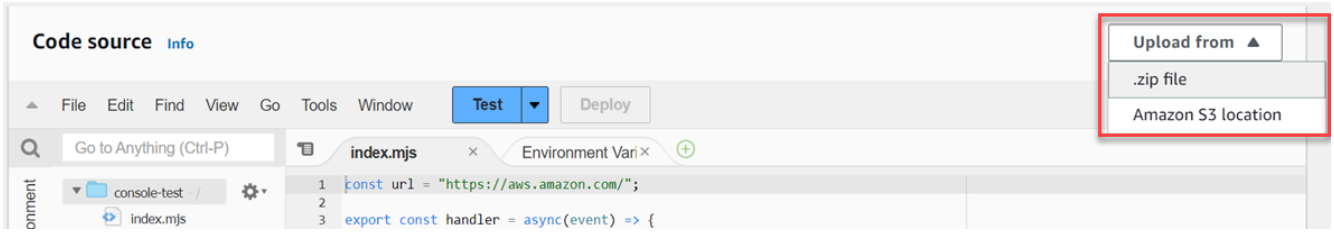
1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih Buat Fungsi Acara.
3. Pilih Author dari awal atau gambar Container.
4. Di bagian Informasi dasar, lakukan hal berikut:
   a. Untuk Nama fungsi, masukkan nama fungsi.
   b. Untuk Runtime, pilih versi bahasa yang akan digunakan untuk fungsi Anda.

5. Perluas Pengaturan lanjutan, lalu pilih Aktifkan enkripsi dengan kunci yang dikelola AWS KMS pelanggan.
6. Memilih kunci yang dikelola pelanggan.
7. Pilih Buat Fungsi Acara.

Untuk menghapus enkripsi kunci terkelola pelanggan, atau menggunakan kunci yang berbeda, Anda harus mengunggah paket deployment .zip lagi.

Untuk menambahkan enkripsi kunci terkelola pelanggan ke fungsi yang ada

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih nama sebuah fungsi.
3. Di panel Sumber kode, pilih Unggah dari.
4. Pilih file.zip atau lokasi Amazon S3.



5. Unggah file atau masukkan lokasi Amazon S3.
6. Pilih Aktifkan enkripsi dengan kunci yang dikelola AWS KMS pelanggan.
7. Memilih kunci yang dikelola pelanggan.
8. Pilih Simpan.

AWS CLI

Untuk menambahkan enkripsi kunci terkelola pelanggan saat Anda membuat fungsi

Dalam contoh create-function berikut:

- `--code`: Menentukan jalur lokal ke paket penyebaran .zip (`ZipFile`) dan kunci yang dikelola pelanggan untuk mengenkripsi (). `SourceKMSKeyArn`
- `--kms-key-arn`: Menentukan kunci yang dikelola pelanggan untuk mengenkripsi variabel lingkungan dan versi paket penyebaran yang tidak di-zip.

```
aws lambda create-function \
  --function-name myFunction \
  --runtime nodejs24.x \
  --handler index.handler \
  --role arn:aws:iam::111122223333:role/service-role/my-lambda-role \
  --code ZipFile=fileb://myFunction.zip,SourceKMSKeyArn=arn:aws:kms:us-
east-1:111122223333:key/key-id \
  --kms-key-arn arn:aws:kms:us-east-1:111122223333:key/key2-id
```

Dalam contoh create-function berikut:

- `--code`: Menentukan lokasi file.zip dalam bucket Amazon S3 `S3Bucket` (`S3Key`,`S3ObjectVersion`,) dan kunci yang dikelola pelanggan untuk mengenkripsinya (). `SourceKMSKeyArn`
- `--kms-key-arn`: Menentukan kunci yang dikelola pelanggan untuk mengenkripsi variabel lingkungan dan versi paket penyebaran yang tidak di-zip.

```
aws lambda create-function \
  --function-name myFunction \
  --runtime nodejs24.x --handler index.handler \
  --role arn:aws:iam::111122223333:role/service-role/my-lambda-role \
  --code S3Bucket=amzn-s3-demo-
bucket,S3Key=myFileName.zip,S3ObjectVersion=myObjectVersion,SourceKMSKeyArn=arn:aws:kms:us-
east-1:111122223333:key/key-id \
  --kms-key-arn arn:aws:kms:us-east-1:111122223333:key/key2-id
```

Untuk menambahkan enkripsi kunci terkelola pelanggan ke fungsi yang ada

Dalam contoh kode fungsi pembaruan-berikut:

- `--zip-file`: Menentukan jalur lokal ke paket.zip deployment.
- `--source-kms-key-arn`: Menentukan kunci yang dikelola pelanggan untuk mengenkripsi versi zip dari paket penyebaran. Lambda menggunakan kunci yang AWS dimiliki untuk mengenkripsi paket unzip untuk pemanggilan fungsi. Jika Anda ingin menggunakan kunci yang dikelola pelanggan untuk mengenkripsi versi paket yang tidak di-zip, jalankan perintah update-function-configuration dengan opsi. `--kms-key-arn`

```
aws lambda update-function-code \
```

```
  --function-name myFunction \
  --zip-file fileb://myFunction.zip \
  --source-kms-key-arn arn:aws:kms:us-east-1:111122223333:key/key-id
```

Dalam contoh kode fungsi pembaruan-berikut:

- `--s3-bucket`: Menentukan lokasi file.zip di bucket Amazon S3.
- `--s3-key`: Menentukan kunci Amazon S3 dari paket penyebaran.
- `--s3-object-version`: Untuk objek berversi, versi objek paket penyebaran yang akan digunakan.
- `--source-kms-key-arn`: Menentukan kunci yang dikelola pelanggan untuk mengenkripsi versi zip dari paket penyebaran. Lambda menggunakan kunci yang AWS dimiliki untuk mengenkripsi paket unzip untuk pemanggilan fungsi. Jika Anda ingin menggunakan kunci yang dikelola pelanggan untuk mengenkripsi versi paket yang tidak di-zip, jalankan perintah update-function-configuration dengan opsi. `--kms-key-arn`

```
aws lambda update-function-code \
  --function-name myFunction \
  --s3-bucket amzn-s3-demo-bucket \
  --s3-key myFileName.zip \
  --s3-object-version myObject Version
  --source-kms-key-arn arn:aws:kms:us-east-1:111122223333:key/key-id
```

Untuk menghapus enkripsi kunci terkelola pelanggan dari fungsi yang ada

Dalam contoh update-function-code berikut, `--zip-file` menentukan jalur lokal ke paket deployment .zip. Saat Anda menjalankan perintah ini tanpa `--source-kms-key-arn` opsi, Lambda menggunakan kunci yang AWS dimiliki untuk mengenkripsi versi zip dari paket penyebaran.

```
aws lambda update-function-code \
  --function-name myFunction \
  --zip-file fileb://myFunction.zip
```

Buat Fungsi Acara Lambda menggunakan gambar kontainer

Kode AWS Lambda fungsi Anda terdiri dari skrip atau program yang dikompilasi dan dependensinya. Gunakan paket deployment untuk men-deploy fungsi kode Anda ke Lambda. Lambda mendukung dua tipe paket deployment: gambar kontainer dan arsip file .zip.

Ada tiga cara untuk membangun image kontainer untuk fungsi Lambda:

- Menggunakan sebuah AWS gambar dasar untuk Lambda

  Gambar AWS dasar dimuat sebelumnya dengan runtime bahasa, klien antarmuka runtime untuk mengelola interaksi antara Lambda dan kode fungsi Anda, dan emulator antarmuka runtime untuk pengujian lokal.
- Menggunakan sebuah AWS OS-only gambar dasar

  AWS OS-only image dasar berisi distribusi Amazon Linux dan emulator antarmuka runtime. Gambar-gambar ini biasanya digunakan untuk membuat gambar kontainer untuk bahasa yang dikompilasi, seperti Go dan Rust, dan untuk versi bahasa atau bahasa yang Lambda tidak menyediakan gambar dasar, seperti Node.js 19. Anda juga dapat menggunakan gambar OS-only dasar untuk mengimplementasikan runtime kustom. Untuk membuat gambar kompatibel dengan Lambda, Anda harus menyertakan klien antarmuka runtime untuk bahasa Anda dalam gambar.
- Menggunakan non-AWS gambar dasar

  Anda dapat menggunakan gambar dasar alternatif dari registri kontainer lain, seperti Alpine Linux atau Debian. Anda juga dapat menggunakan gambar kustom yang dibuat oleh organisasi Anda. Untuk membuat gambar kompatibel dengan Lambda, Anda harus menyertakan klien antarmuka runtime untuk bahasa Anda dalam gambar.

> **Tip**
>
> Untuk mengurangi waktu yang dibutuhkan agar fungsi penampung Lambda menjadi aktif, lihat Menggunakan build multi-tahap dalam dokumentasi Docker. Untuk membuat gambar kontainer yang efisien, ikuti Praktik terbaik untuk menulis Dockerfiles.

Untuk membuat fungsi Lambda dari image kontainer, buat gambar Anda secara lokal dan unggah ke repositori Amazon Elastic Container Registry (Amazon ECR). Jika Anda menggunakan gambar kontainer yang disediakan oleh AWS Marketplacepenjual, Anda harus mengkloning gambar ke

repositori Amazon ECR pribadi Anda terlebih dahulu. Kemudian, tentukan URI repositori saat Anda membuat fungsi. Repositori Amazon ECR harus sama Wilayah AWS dengan fungsi Lambda. Anda dapat membuat fungsi menggunakan gambar di AWS akun yang berbeda, selama gambar berada di Wilayah yang sama dengan fungsi Lambda. Untuk informasi selengkapnya, lihat Izin lintas akun Amazon ECR.

> **Note**
>
> Lambda tidak mendukung titik akhir Amazon ECR FIPS untuk gambar kontainer. Jika URI repositori Anda termasuk`ecr-fips`, Anda menggunakan titik akhir FIPS. Contoh:`111122223333.dkr.ecr-fips.us-east-1.amazonaws.com`.

Halaman ini menjelaskan jenis gambar dasar dan persyaratan untuk membuat gambar Lambda-compatible kontainer.

> **Note**
>
> Anda tidak dapat mengubah jenis paket penerapan (.zip atau image kontainer) untuk fungsi yang ada. Misalnya, Anda tidak dapat mengonversi fungsi gambar kontainer untuk menggunakan arsip file.zip. Anda harus membuat fungsi baru.

Topik

- Persyaratan
- Menggunakan sebuah AWS gambar dasar untuk Lambda
- Menggunakan sebuah AWS OS-only gambar dasar
- Menggunakan non-AWS gambar dasar
- Klien antarmuka runtime
- Izin Amazon ECR
- Siklus hidup fungsi

Persyaratan

Instal AWS CLI versi 2 dan CLI Docker. Selain itu, perhatikan persyaratan berikut:

- Gambar kontainer harus mengimplementasikan fileMenggunakan API runtime Lambda untuk runtime kustom. Klien antarmuka runtime sumber terbuka AWS menerapkan API. Anda dapat menambahkan klien antarmuka runtime untuk gambar dasar pilihan Anda untuk membuatnya kompatibel dengan Lambda.
- Gambar kontainer harus dapat berjalan pada sistem file hanya baca. Kode fungsi Anda dapat mengakses `/tmp` direktori yang dapat ditulis dengan penyimpanan antara 512 MB dan 10.240 MB, dengan penambahan 1-MB, penyimpanan.
- Pengguna Lambda default harus dapat membaca semua file yang diperlukan untuk menjalankan kode fungsi Anda. Lambda mengikuti praktik terbaik keamanan dengan mendefinisikan pengguna Linux default dengan izin yang paling tidak istimewa. Ini berarti Anda tidak perlu menentukan USER di Dockerfile Anda. Verifikasi bahwa kode aplikasi Anda tidak bergantung pada file yang dibatasi pengguna Linux lain untuk berjalan.
- Lambda hanya mendukung gambar Linux-based kontainer.
- Lambda menyediakan gambar dasar multi-arsitektur. Namun, gambar yang Anda buat untuk fungsi Anda harus menargetkan hanya satu arsitektur. Lambda tidak mendukung fungsi yang menggunakan gambar kontainer multi-arsitektur.

Menggunakan sebuah AWS gambar dasar untuk Lambda

Anda dapat menggunakan salah satu gambar AWS dasar untuk Lambda untuk membangun gambar kontainer untuk kode fungsi Anda. Gambar dasar yang dimuat sebelumnya dengan runtime bahasa dan komponen lain yang diperlukan untuk menjalankan gambar kontainer pada Lambda. Anda menambahkan kode fungsi dan dependensi ke gambar dasar, lalu mengemasnya sebagai gambar kontainer.

AWS secara berkala memberikan pembaruan pada gambar AWS dasar untuk Lambda. Jika Dockerfile Anda menyertakan nama gambar di properti FROM, klien Docker Anda akan menarik versi terbaru gambar dari repositori Amazon ECR. Untuk menggunakan gambar dasar diperbarui, Anda harus membangun kembali gambar kontainer Anda dan memperbarui kode fungsi.

Gambar dasar Node.js 20, Python 3.12, Java 21, .NET 8, Ruby 3.3, dan yang lebih baru didasarkan pada gambar kontainer minimal Amazon Linux 2023. Gambar dasar sebelumnya menggunakan Amazon Linux 2. AL2023 memberikan beberapa keunggulan dibandingkan Amazon Linux 2, termasuk jejak penyebaran yang lebih kecil dan versi pustaka yang diperbarui seperti. `glibc`

AL2023-based gambar menggunakan `microdnf` (symlinked as`dnf`) sebagai manajer paket, bukan`yum`, yang merupakan manajer paket default di Amazon Linux 2. `microdnf`adalah

implementasi mandiri dari. `dnf` Untuk daftar paket yang disertakan dalam AL2023-based gambar, lihat kolom Penampung Minimal di Membandingkan paket yang diinstal di Amazon Linux 2023 Container Images. Untuk informasi selengkapnya tentang perbedaan antara AL2023 dan Amazon Linux 2, lihat Memperkenalkan AWS Lambda runtime Amazon Linux 2023 untuk di Blog Komputasi. AWS

> **Note**
>
> Untuk menjalankan AL2023-based gambar secara lokal, termasuk with AWS Serverless Application Model (AWS SAM), Anda harus menggunakan Docker versi 20.10.10 atau yang lebih baru.

Untuk membuat gambar kontainer menggunakan gambar AWS dasar, pilih instruksi untuk bahasa pilihan Anda:

- Node.js
- TypeScript(menggunakan gambar Node.js dasar)
- Python
- Java
- Go
- .NET
- Ruby

Menggunakan sebuah AWS OS-only gambar dasar

AWS OS-only image dasar berisi distribusi Amazon Linux dan emulator antarmuka runtime. Gambar-gambar ini biasanya digunakan untuk membuat gambar kontainer untuk bahasa yang dikompilasi, seperti Go dan Rust, dan untuk versi bahasa atau bahasa yang Lambda tidak menyediakan gambar dasar, seperti Node.js 19. Anda juga dapat menggunakan gambar OS-only dasar untuk mengimplementasikan runtime kustom. Untuk membuat gambar kompatibel dengan Lambda, Anda harus menyertakan klien antarmuka runtime untuk bahasa Anda dalam gambar.

Tanda	Waktu berjalan	Sistem operasi	Dockerfile	penghentian
al2023	OS-only Runtime	Amazon Linux 2023	Dockerfile untuk OS-only Runtime aktif GitHub	30 Jun 2029

Galeri Publik Registri Kontainer Elastis Amazon: gallery.ecr. aws/lambda/disediakan

Menggunakan non-AWS gambar dasar

Lambda mendukung gambar apa pun yang sesuai dengan salah satu format manifes gambar berikut:

- Docker Image Manifest V2, skema 2 (digunakan dengan Docker versi 1.10 dan yang lebih baru)
- Spesifikasi Open Container Initiative (OCI) (v1.0.0 dan yang lebih tinggi)

Lambda mendukung ukuran gambar maksimum yang tidak terkompresi 10 GB, termasuk semua lapisan.

> **Note**
>
> - Untuk membuat gambar kompatibel dengan Lambda, Anda harus menyertakan klien antarmuka runtime untuk bahasa Anda dalam gambar.
> - Untuk kinerja optimal, pertahankan ukuran manifes gambar Anda di bawah 25.400 byte. Untuk mengurangi ukuran manifes gambar, minimalkan jumlah lapisan dalam gambar Anda dan kurangi anotasi.

Klien antarmuka runtime

Jika Anda menggunakan gambar OS-only dasar atau gambar dasar alternatif, Anda harus menyertakan klien antarmuka runtime dalam gambar Anda. Klien antarmuka runtime harus memperluasMenggunakan API runtime Lambda untuk runtime kustom, yang mengelola interaksi antara Lambda dan kode fungsi Anda. AWS menyediakan klien antarmuka runtime sumber terbuka untuk bahasa berikut:

- Node.js

- Python
- Java
- .NET
- Go
- Ruby
- Karat —

Jika Anda menggunakan bahasa yang tidak memiliki klien antarmuka runtime AWS yang disediakan, Anda harus membuatnya sendiri.

Izin Amazon ECR

Sebelum Anda membuat fungsi Lambda dari gambar kontainer, Anda harus membangun gambar secara lokal dan mengunggahnya ke repositori Amazon ECR. Saat Anda membuat fungsi, tentukan URI repositori Amazon ECR.

Pastikan bahwa izin untuk pengguna atau peran yang membuat fungsi mencakup`GetRepositoryPolicy`,, `SetRepositoryPolicy`, `BatchGetImage`, dan`GetDownloadUrlForLayer`.

Untuk contoh, gunakan konsol IAM untuk membuat peran dengan kebijakan berikut:

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": [
        "ecr:SetRepositoryPolicy",
        "ecr:GetRepositoryPolicy",
        "ecr:BatchGetImage",
        "ecr:GetDownloadUrlForLayer"
      ],
      "Resource": "arn:aws:ecr:us-east-1:111122223333:repository/hello-world"
    }
```

```
    ]
}
```

Izin yang dibutuhkan Lambda untuk mengambil gambar penampung Anda bergantung pada apakah repositori Amazon ECR berada di akun yang AWS sama dengan fungsi atau di akun yang berbeda. Bagian berikut menjelaskan persyaratan untuk setiap skenario.

> **Note**
>
> Di IAM, akses akun yang sama hanya memerlukan satu sisi untuk memberikan izin — baik kebijakan berbasis identitas (pada peran) atau kebijakan berbasis sumber daya (di repositori Amazon ECR). Cross-account akses mengharuskan kedua belah pihak untuk memberikan izin — kebijakan berbasis identitas tentang peran dalam akun konsumsi DAN kebijakan berbasis sumber daya pada repositori Amazon ECR di akun pemilik harus mengizinkan tindakan tersebut.

Same-account Kebijakan repositori Amazon ECR

Untuk fungsi di akun yang sama dengan image container di Amazon ECR, Anda dapat memberikan Lambda akses melalui kebijakan berbasis identitas tentang peran eksekusi atau kebijakan berbasis sumber daya di repositori Amazon ECR. Hanya satu pihak yang perlu mengizinkan akses.

Jika Anda memilih untuk menggunakan kebijakan repositori Amazon ECR, tambahkan `ecr:BatchGetImage` dan izin. `ecr:GetDownloadUrlForLayer` Contoh berikut menunjukkan kebijakan minimum:

```
{
        "Sid": "LambdaECRImageRetrievalPolicy",
        "Effect": "Allow",
        "Principal": {
          "Service": "lambda.amazonaws.com"
        },
        "Action": [
          "ecr:BatchGetImage",
          "ecr:GetDownloadUrlForLayer"
        ]
    }
```

Untuk informasi selengkapnya tentang izin repositori Amazon ECR, lihat Kebijakan repositori pribadi di Panduan Pengguna Amazon Elastic Container Registry.

Jika repositori Amazon ECR tidak menyertakan izin ini, Lambda mencoba menambahkannya secara otomatis. Lambda dapat menambahkan izin hanya jika pemanggil utama Lambda memiliki dan izin. `ecr:getRepositoryPolicy` `ecr:setRepositoryPolicy`

Untuk melihat atau mengedit izin repositori Amazon ECR Anda, ikuti petunjuk dalam Menyetel pernyataan kebijakan repositori pribadi di Panduan Pengguna Amazon Elastic Container Registry.

Izin lintas akun Amazon ECR

Ketika fungsi dalam satu akun menggunakan gambar kontainer dari repositori Amazon ECR di akun yang berbeda, kedua belah pihak harus memberikan akses. Kebijakan berbasis identitas tentang peran dalam akun konsumsi harus mengizinkan `ecr:BatchGetImage` dan`ecr:GetDownloadUrlForLayer`, dan kebijakan berbasis sumber daya pada repositori Amazon ECR di akun pemilik juga harus mengizinkan tindakan ini.

Dalam contoh berikut, kebijakan izin repositori Amazon ECR Anda memerlukan pernyataan berikut untuk memberikan akses ke nomor akun 123456789012.

- CrossAccountPermission— Memungkinkan akun 123456789012 untuk membuat dan memperbarui fungsi Lambda yang menggunakan gambar dari repositori ECR ini.
- LambdaECRImageCrossAccountRetrievalPolicy— Lambda akhirnya menetapkan status fungsi menjadi tidak aktif jika tidak dipanggil untuk waktu yang lama. Pernyataan ini diperlukan agar Lambda dapat mengambil gambar kontainer untuk optimasi dan caching atas nama fungsi yang dimiliki oleh 123456789012.

Example— Tambahkan izin lintas akun ke repositori Anda

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "CrossAccountPermission",
      "Effect": "Allow",
      "Action": [
```

```
        "ecr:BatchGetImage",
        "ecr:GetDownloadUrlForLayer"
      ],
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:root"
      },
      "Resource": "arn:aws:ecr:us-east-1:123456789012:repository/example-lambda-
repository"
    },
    {
      "Sid": "LambdaECRImageCrossAccountRetrievalPolicy",
      "Effect": "Allow",
      "Action": [
        "ecr:BatchGetImage",
        "ecr:GetDownloadUrlForLayer"
      ],
      "Principal": {
        "Service": "lambda.amazonaws.com"
      },
      "Condition": {
        "ArnLike": {
          "aws:sourceARN": "arn:aws:lambda:us-east-1:123456789012:function:*"
        }
      },
      "Resource": "arn:aws:ecr:us-east-1:123456789012:repository/example-lambda-
repository"
    }
  ]
}
```

Untuk memberikan akses ke beberapa akun, Anda menambahkan ID akun ke daftar Utama dalam `CrossAccountPermission` kebijakan dan daftar Evaluasi kondisi di`LambdaECRImageCrossAccountRetrievalPolicy`.

Jika Anda bekerja dengan beberapa akun di AWS Organisasi, sebaiknya Anda menghitung setiap ID akun dalam kebijakan izin ECR. Pendekatan ini sejalan dengan praktik terbaik AWS keamanan dalam menyetel izin sempit dalam kebijakan IAM.

Selain kebijakan repositori Amazon ECR, pengguna atau peran yang membuat fungsi juga harus memiliki `BatchGetImage` dan `GetDownloadUrlForLayer` izin dalam kebijakan berbasis identitas mereka.

Siklus hidup fungsi

Setelah Anda mengunggah gambar kontainer baru atau yang diperbarui, Lambda mengoptimalkan gambar sebelum fungsi dapat memproses pemanggilan. Proses pengoptimalan dapat memakan waktu beberapa detik. Fungsi tetap dalam `Pending` keadaan sampai proses selesai, ketika keadaan transisi ke. `Active` Anda tidak dapat memanggil fungsi sampai mencapai `Active` status.

Jika fungsi tidak dipanggil selama beberapa minggu, Lambda mengeklaim kembali versi dioptimalkan, dan fungsi bertransisi ke status `Inactive`. Untuk mengaktifkan kembali fungsinya, Anda harus memanggilnya. Lambda menolak invokasi pertama dan fungsi memasuki status `Pending` sampai Lambda mengoptimalkan kembali gambar. Fungsi kemudian kembali ke status `Active`.

Lambda secara berkala mengambil gambar kontainer terkait dari repositori Amazon ECR. Jika image kontainer yang sesuai tidak ada lagi di Amazon ECR atau izin dicabut, fungsi akan memasuki `Failed` status, dan Lambda mengembalikan kegagalan untuk pemanggilan fungsi apa pun.

Anda dapat menggunakan API Lambda untuk mendapatkan informasi tentang status fungsi. Lihat informasi yang lebih lengkap di Status Fungsi Acara Lambda.

Self-managed Penyimpanan kode S3

Secara default, saat Anda membuat atau memperbarui Fungsi Peristiwa Lambda atau lapisan dari arsip file.zip, Lambda menyimpan salinan kode Anda dalam penyimpanan. Lambda-managed Setiap AWS akun memiliki kuota 300 GB untuk Lambda-managed penyimpanan per Wilayah.

Dengan penyimpanan kode S3 yang dikelola sendiri, Anda dapat mengonfigurasi Lambda untuk mereferensikan kode Anda langsung dari bucket S3 di akun Anda. Lambda tidak menyimpan salinan kode Anda, sehingga kode tidak dihitung terhadap kuota Lambda-managed penyimpanan Anda. Lambda mengakses kode Anda langsung dari bucket S3 Anda.

Self-managed Penyimpanan kode S3 tersedia untuk semua fungsi dan lapisan yang dibuat dan diperbarui menggunakan arsip file.zip. Ini kompatibel dengan semua kelas penyimpanan S3 kecuali kelas penyimpanan Glacier.

> **Note**
>
> Self-managed Penyimpanan kode S3 tidak mengubah ukuran paket.zip deployment maksimum, yang tetap pada 250 MB (unzip).

Topik

- Menyiapkan penyimpanan kode S3 yang dikelola sendiri
- Mengonfigurasi izin
- Membuat fungsi dengan penyimpanan S3 yang dikelola sendiri
- Menggunakan penyimpanan S3 yang dikelola sendiri dengan lapisan
- Beralih di antara mode penyimpanan
- Siklus hidup fungsi dengan penyimpanan yang dikelola sendiri
- Cross-account dan penggunaan lintas wilayah
- Pertimbangan-pertimbangan

Menyiapkan penyimpanan kode S3 yang dikelola sendiri

Untuk menggunakan penyimpanan kode S3 yang dikelola sendiri, selesaikan langkah-langkah berikut:

1. Buat bucket S3 atau gunakan bucket yang ada di akun Anda.
2. Aktifkan versi S3 untuk bucket. Lambda memerlukan versi S3 untuk melacak versi objek sumber Anda yang akan digunakan. Setiap kali Anda mengunggah paket.zip baru, S3 membuat versi baru sehingga Lambda dapat membedakan pembaruan.
3. Unggah paket penerapan.zip Anda ke bucket S3.
4. Berikan prinsip layanan Lambda `s3:GetObject` dan `s3:GetObjectVersion` izin untuk objek sumber. Lihat Mengonfigurasi izin.
5. Buat atau perbarui fungsi atau lapisan Anda, tentukan objek S3, tombol S3, versi objek S3, dan pengaturan ke. `S3ObjectStorageMode` `REFERENCE`

Mengonfigurasi izin

Untuk menggunakan penyimpanan kode S3 yang dikelola sendiri, berikan izin Lambda service principal (`lambda.amazonaws.com`) untuk mengakses objek sumber Anda. Tambahkan kebijakan bucket S3 berikut:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "LambdaSelfManagedCodeAccess",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:GetObjectVersion"
      ],
      "Resource": [
        "arn:aws:s3:::my-bucket/my-function.zip"
      ],
      "Principal": {
        "Service": "lambda.amazonaws.com"
      },
      "Condition": {
        "ArnLike": {
          "aws:SourceArn": "arn:aws:lambda:us-east-1:111122223333:function:my-function"
        }
      }
    }
  ]
```

```
}
```

Kebijakan bucket S3 ini terpisah dari peran eksekusi fungsi Anda. Kebijakan bucket adalah kebijakan berbasis sumber daya yang memberikan izin kepada Lambda service principal (`lambda.amazonaws.com`) untuk mengambil objek kode tertentu sehingga Lambda dapat mengoptimalkan fungsi Anda. Itu tidak memberikan izin kepada prinsipal lain, dan tidak memberikan izin untuk memanggil fungsi Anda. Peran eksekusi fungsi Anda adalah identitas IAM terpisah yang diasumsikan Lambda saat menjalankan fungsi Anda; Anda memberikan izin pemanggilan dan runtime lainnya melalui kebijakan IAM yang dilampirkan pada peran tersebut. Memberikan akses utama layanan Lambda ke objek di bucket Anda tidak mengizinkan identitas lain mengakses objek tersebut atau menjalankan fungsi Anda.

Membuat fungsi dengan penyimpanan S3 yang dikelola sendiri

Anda dapat membuat fungsi dengan penyimpanan kode S3 yang dikelola sendiri menggunakan konsol,, atau. AWS CLI AWS CloudFormation

> **Note**
>
> Editor kode konsol Lambda tidak tersedia untuk fungsi yang menggunakan mode Referensi. Untuk mengedit kode fungsi, perbarui file.zip di bucket S3 Anda dan perbarui fungsinya.

Menggunakan konsol

Saat Anda memperbarui fungsi dari lokasi S3 di konsol Lambda, Anda dapat memilih mode penyimpanan kode.

Untuk memperbarui fungsi untuk menggunakan mode Referensi (konsol)

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih fungsi yang akan diperbarui dan pilih tab Kode.
3. Di bawah Sumber kode, pilih Perbarui, lalu pilih Perbarui dari file di Amazon S3.
4. Untuk URL tautan Amazon S3, masukkan URL tautan S3 dari file.zip Anda.
5. Di bawah Mode penyimpanan kode, pilih Mode referensi.
6. Pilih Simpan.

Setelah Anda mengonfigurasi mode Referensi, bagian Properti kode pada tab Kode fungsi menampilkan mode penyimpanan Kode: Referensi dan lokasi S3 yang menunjuk ke bucket S3 Anda.

Menggunakan AWS CLI

Gunakan `create-function` perintah dengan `S3ObjectStorageMode=REFERENCE` parameter:

```
aws lambda create-function \
  --function-name my-function \
  --runtime python3.12 \
  --role arn:aws:iam::111122223333:role/lambda-execution-role \
  --handler lambda_function.lambda_handler \
  --code S3Bucket=my-bucket,\
S3Key=my-function.zip,\
S3ObjectVersion=abc123def456,\
S3ObjectStorageMode=REFERENCE
```

Untuk memperbarui fungsi yang ada untuk menggunakan penyimpanan S3 yang dikelola sendiri:

```
aws lambda update-function-code \
  --function-name my-function \
  --s3-bucket my-bucket \
  --s3-key my-function.zip \
  --s3-object-version abc123def456 \
  --s3-object-storage-mode REFERENCE
```

> **Important**
>
> Anda harus menentukan `S3ObjectStorageMode=REFERENCE` pada setiap panggilan keu`pdate-function-code`. Jika Anda menghilangkan`S3ObjectStorageMode`, defaultnya dan `COPY` Lambda menyimpan kode Anda dalam penyimpanan. Lambda-managed

Penggunaan AWS CloudFormation

Dalam CloudFormation template Anda, atur `S3ObjectStorageMode` ke `REFERENCE` dalam `Code` properti `AWS::Lambda::Function` sumber daya:

```
Resources:
```

```
  MyFunction:
    Type: AWS::Lambda::Function
    Properties:
      FunctionName: my-function
      Runtime: python3.12
      Handler: lambda_function.lambda_handler
      Role: !GetAtt LambdaExecutionRole.Arn
      Code:
        S3Bucket: my-bucket
        S3Key: my-function.zip
        S3ObjectVersion: abc123def456
        S3ObjectStorageMode: REFERENCE
```

Menggunakan penyimpanan S3 yang dikelola sendiri dengan lapisan

Anda juga dapat menggunakan penyimpanan kode S3 yang dikelola sendiri untuk versi lapisan. Gunakan `publish-layer-version` perintah dengan`S3ObjectStorageMode=REFERENCE`:

```
aws lambda publish-layer-version \
  --layer-name my-layer \
  --content S3Bucket=my-bucket,S3Key=my-layer.zip,S3ObjectStorageMode=REFERENCE \
  --compatible-runtimes python3.12
```

Anda dapat menggunakan kombinasi penyimpanan yang Lambda-managed dikelola sendiri untuk kode fungsi dan kode lapisan.

Beralih di antara mode penyimpanan

Anda dapat beralih Lambda-managed antara penyimpanan kode S3 yang dikelola sendiri saat memperbarui kode fungsi atau menerbitkan versi lapisan baru. Setel `S3ObjectStorageMode` ke salah satu `COPY` atau`REFERENCE`:

- `COPY`(default) — Lambda menyalin kode sumber Anda ke Lambda-managed penyimpanan.
- `REFERENCE`— Lambda mereferensikan kode sumber Anda langsung dari bucket S3 Anda tanpa menyimpan salinannya.

Saat Anda beralih dari `COPY` ke`REFERENCE`, Lambda menghapus Lambda-managed salinan kode sumber Anda. Saat Anda beralih dari `REFERENCE` ke`COPY`, Lambda membuat salinan kode sumber Anda di Lambda-managed penyimpanan. Beralih ke `COPY` gagal jika Anda melebihi batas penyimpanan Lambda-managed kode.

Siklus hidup fungsi dengan penyimpanan yang dikelola sendiri

Lambda secara berkala mengakses objek sumber dari bucket S3 Anda untuk mengoptimalkan ulang kode fungsi Anda. Anda harus mempertahankan akses ke objek sumber agar fungsi Anda tetap aktif.

- Jika Lambda kehilangan akses ke objek sumber untuk suatu fungsi, fungsi bertransisi ke status. `Inactive` Untuk mengembalikan fungsi, kembalikan akses ke objek sumber dan perbarui fungsinya.
- Jika Lambda kehilangan akses ke objek sumber untuk lapisan, fungsinya tetap ada. `Active` Anda dapat memperbarui konfigurasi fungsi (tanpa memperbarui lapisan), memperbarui kode fungsi, atau memperbarui konfigurasi fungsi dengan lapisan baru. Namun, memperbarui konfigurasi fungsi dengan lapisan yang tidak dapat diakses gagal.

Cross-account dan penggunaan lintas wilayah

Anda dapat membuat fungsi dan lapisan dari kode sumber di bucket S3 di akun yang berbeda dan di Wilayah yang berbeda. Cross-Region pengambilan objek menimbulkan biaya transfer data S3 tambahan. Lihat harga S3 untuk detailnya.

Pertimbangan-pertimbangan

- Self-managed Penyimpanan kode S3 tersedia untuk fungsi dan lapisan arsip file.zip saja. Fungsi gambar kontainer terus menggunakan Amazon ECR.
- Ukuran paket.zip deployment maksimum (250 MB unzip) tidak berubah.
- Anda membayar tarif penyimpanan Amazon S3 standar untuk kode yang disimpan dalam bucket yang dikelola sendiri. Lambda tidak mengenakan biaya untuk pengambilan objek S3 dalam Wilayah yang sama.
- Self-managed Penyimpanan kode S3 kompatibel dengan semua kelas penyimpanan S3 kecuali kelas penyimpanan Glacier.
- Anda dapat menggunakan fitur S3 seperti Cross-Region Replikasi S3 untuk berbagi artefak di seluruh wilayah dan kebijakan siklus hidup S3 untuk mengelola objek.

Konfigurasikan memori Fungsi Peristiwa Lambda

Lambda memberikan daya CPU sebanding dengan jumlah memori yang dikonfigurasi. Memori adalah jumlah memori yang tersedia untuk fungsi Lambda saat waktu aktif. Anda dapat menambah atau mengurangi memori dan daya CPU yang dialokasikan ke fungsi Anda menggunakan pengaturan Memori. Anda dapat mengonfigurasi memori antara 128 MB dan 10.240 MB dengan peningkatan 1-MB. Pada 1.769 MB, fungsi memiliki setara dengan satu vCPU (satu v CPU-second kredit per detik).

Halaman ini menjelaskan bagaimana dan kapan harus memperbarui pengaturan memori untuk fungsi Lambda.

Bagian-bagian

- Menentukan pengaturan memori yang sesuai untuk fungsi Lambda
- Mengonfigurasi memori fungsi (konsol)
- Mengkonfigurasi memori fungsi (AWS CLI)
- Mengkonfigurasi memori fungsi (AWS SAM)
- Menerima rekomendasi memori fungsi (konsol)

Menentukan pengaturan memori yang sesuai untuk fungsi Lambda

Memori adalah tuas utama untuk mengontrol kinerja suatu fungsi. Pengaturan default, 128 MB, adalah pengaturan serendah mungkin. Kami menyarankan Anda hanya menggunakan 128 MB untuk fungsi Lambda sederhana, seperti yang mengubah dan merutekan acara ke layanan lain AWS .

Alokasi memori yang lebih tinggi dapat meningkatkan kinerja untuk fungsi yang menggunakan pustaka impor, lapisan Lambda, Amazon Simple Storage Service (Amazon S3) atau Amazon Elastic File System (Amazon EFS). Menambahkan lebih banyak memori secara proporsional meningkatkan jumlah CPU, meningkatkan daya komputasi keseluruhan yang tersedia. Jika suatu fungsi adalah CPU, jaringan atau terikat memori, maka meningkatkan pengaturan memori dapat secara dramatis meningkatkan kinerjanya.

Untuk menemukan konfigurasi memori yang tepat, pantau fungsi Anda dengan Amazon CloudWatch dan atur alarm jika konsumsi memori mendekati maksimum yang dikonfigurasi. Ini dapat membantu mengidentifikasi fungsi terikat memori. Untuk CPU-bound dan IO-bound fungsi, pemantauan durasi juga dapat memberikan wawasan. Dalam kasus ini, meningkatkan memori dapat membantu menyelesaikan kemacetan komputasi atau jaringan.

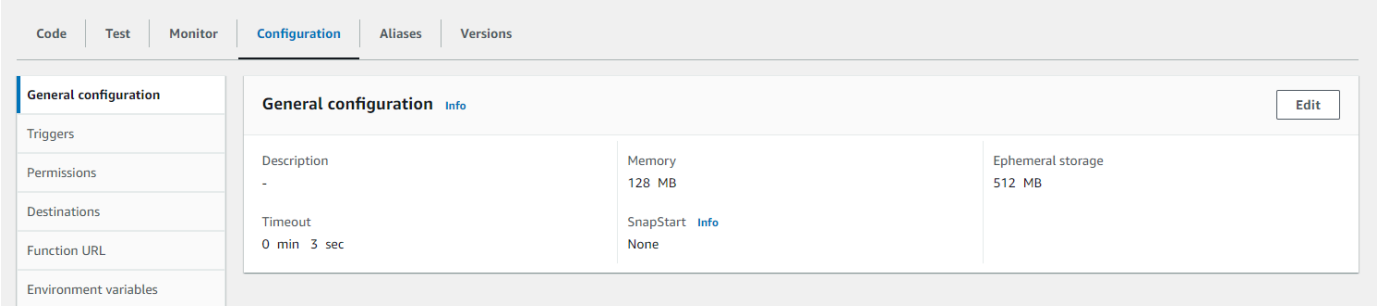
Anda juga dapat mempertimbangkan untuk menggunakan alat AWS Lambda Power Tuning open source. Alat ini digunakan AWS Step Functions untuk menjalankan beberapa versi bersamaan dari fungsi Lambda pada alokasi memori yang berbeda dan mengukur kinerja. Fungsi input berjalan di AWS akun Anda, melakukan panggilan HTTP langsung dan interaksi SDK, untuk mengukur kemungkinan kinerja dalam skenario produksi langsung. Anda juga dapat menerapkan CI/CD proses untuk menggunakan alat ini untuk secara otomatis mengukur kinerja fungsi baru yang Anda terapkan.

Mengonfigurasi memori fungsi (konsol)

Anda dapat mengonfigurasi memori fungsi Anda di konsol Lambda.

Untuk memperbarui memori fungsi

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih fungsi.
3. Pilih tab Konfigurasi dan kemudian pilih Konfigurasi umum.



4. Di bagian Konfigurasi umum, pilih Edit.
5. Untuk Memori, tetapkan nilai dari 128 MB menjadi 10.240 MB.
6. Pilih Simpan.

Mengkonfigurasi memori fungsi (AWS CLI)

Anda dapat menggunakan perintah update-function-configuration untuk mengkonfigurasi memori fungsi Anda.

Example

```
aws lambda update-function-configuration \
  --function-name my-function \
  --memory-size 1024
```

Mengkonfigurasi memori fungsi (AWS SAM)

Anda dapat menggunakan AWS Serverless Application Modeluntuk mengkonfigurasi memori untuk fungsi Anda. Perbarui MemorySizeproperti di `template.yaml` file Anda dan kemudian jalankan sam deploy.

Example template.yaml

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: An AWS Serverless Application Model template describing your function.
Resources:
  my-function:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: .
      Description: ''
      MemorySize: 1024
      # Other function properties...
```

Menerima rekomendasi memori fungsi (konsol)

Jika Anda memiliki izin administrator di AWS Identity and Access Management (IAM), Anda dapat memilih untuk menerima rekomendasi pengaturan memori fungsi Lambda dari. AWS Compute Optimizer Untuk petunjuk cara memilih rekomendasi memori untuk akun atau organisasi Anda, lihat Memilih akun Anda dalam Panduan Pengguna AWS Compute Optimizer .

> **Note**
>
> Compute Optimizer hanya mendukung fungsi yang menggunakan arsitektur x86_64.

Ketika Anda telah memilih dan fungsi Lambda Anda memenuhi persyaratan Compute Optimizer, Anda dapat melihat dan menerima rekomendasi memori fungsi dari Compute Optimizer di konsol Lambda dalam konfigurasi Umum.

Konfigurasikan penyimpanan singkat untuk Fungsi Acara Lambda

Lambda menyediakan penyimpanan sementara untuk fungsi dalam direktori. `/tmp` Penyimpanan ini bersifat sementara dan unik untuk setiap lingkungan eksekusi. Anda dapat mengontrol jumlah penyimpanan sementara yang dialokasikan ke fungsi Anda menggunakan pengaturan penyimpanan Ephemeral. Anda dapat mengonfigurasi penyimpanan sementara antara 512 MB dan 10.240 MB, dengan peningkatan 1-MB. Semua data yang disimpan `/tmp` dienkripsi saat istirahat dengan kunci yang dikelola oleh. AWS

Halaman ini menjelaskan kasus penggunaan umum dan cara memperbarui penyimpanan sementara untuk fungsi Lambda.

Bagian-bagian

- Kasus penggunaan umum untuk peningkatan penyimpanan sementara
- Mengkonfigurasi penyimpanan sementara (konsol)
- Mengkonfigurasi penyimpanan sementara (AWS CLI)
- Mengkonfigurasi penyimpanan sementara (AWS SAM)

Kasus penggunaan umum untuk peningkatan penyimpanan sementara

Berikut adalah beberapa kasus penggunaan umum yang mendapat manfaat dari peningkatan penyimpanan sementara:

- Extract-transform-load Pekerjaan (ETL): Tingkatkan penyimpanan sementara saat kode Anda melakukan komputasi menengah atau mengunduh sumber daya lain untuk menyelesaikan pemrosesan. Lebih banyak ruang sementara memungkinkan pekerjaan ETL yang lebih kompleks untuk berjalan di fungsi Lambda.
- Inferensi pembelajaran mesin (ML): Banyak tugas inferensi bergantung pada file data referensi besar, termasuk perpustakaan dan model. Dengan penyimpanan yang lebih singkat, Anda dapat mengunduh model yang lebih besar dari Amazon Simple Storage Service (Amazon S3) `/tmp` ke dan menggunakannya dalam pemrosesan Anda.
- Pemrosesan data: Untuk beban kerja yang mengunduh objek dari Amazon S3 sebagai respons terhadap peristiwa S3, `/tmp` lebih banyak ruang memungkinkan untuk menangani objek yang lebih besar tanpa menggunakan pemrosesan dalam memori. Beban kerja yang membuat PDF atau media proses juga mendapat manfaat dari penyimpanan yang lebih singkat.

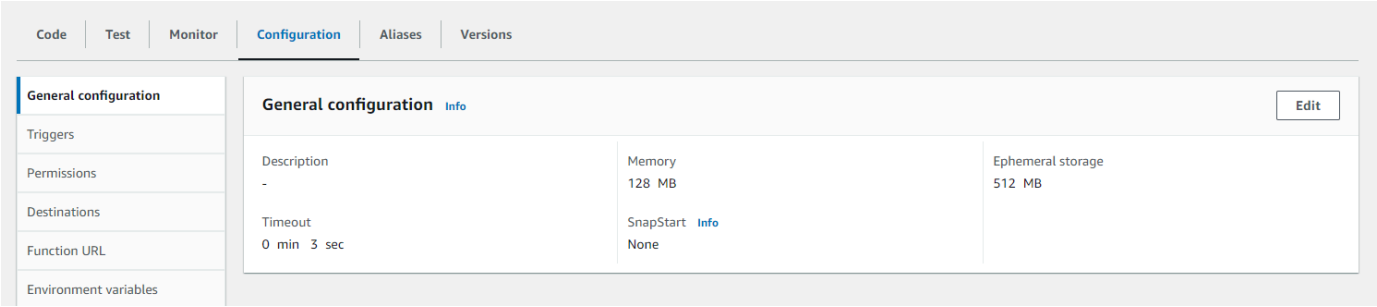
- Pemrosesan grafis: Pemrosesan gambar adalah kasus penggunaan umum untuk Lambda-based aplikasi. Untuk beban kerja yang memproses file TIFF besar atau citra satelit, penyimpanan yang lebih singkat memudahkan penggunaan pustaka dan melakukan perhitungan di Lambda.

Mengkonfigurasi penyimpanan sementara (konsol)

Anda dapat mengonfigurasi penyimpanan sementara di konsol Lambda.

Untuk memodifikasi penyimpanan sementara untuk suatu fungsi

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih fungsi.
3. Pilih tab Konfigurasi dan kemudian pilih Konfigurasi umum.



4. Di bagian Konfigurasi umum, pilih Edit.
5. Untuk penyimpanan Ephemeral, tetapkan nilai antara 512 MB dan 10.240 MB, dengan peningkatan 1-MB.
6. Pilih Simpan.

Mengkonfigurasi penyimpanan sementara (AWS CLI)

Anda dapat menggunakan perintah update-function-configuration untuk mengkonfigurasi penyimpanan sementara.

Example

```
aws lambda update-function-configuration \
  --function-name my-function \
  --ephemeral-storage '{"Size": 1024}'
```

Mengkonfigurasi penyimpanan sementara (AWS SAM)

Anda dapat menggunakan AWS Serverless Application Modeluntuk mengkonfigurasi penyimpanan sementara untuk fungsi Anda. Perbarui EphemeralStorageproperti di `template.yaml` file Anda dan kemudian jalankan sam deploy.

Example template.yaml

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: An AWS Serverless Application Model template describing your function.
Resources:
  my-function:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: .
      Description: ''
      MemorySize: 128
      Timeout: 120
      Handler: index.handler
      Runtime: nodejs22.x
      Architectures:
        - x86_64
      EphemeralStorage:
        Size: 10240
      # Other function properties...
```

Memilih dan mengonfigurasi arsitektur set instruksi untuk fungsi Lambda Anda

Yang arsitektur set instruksi Fungsi Lambda menentukan jenis prosesor komputer yang digunakan Lambda untuk menjalankan fungsi tersebut. Lambda menyediakan pilihan arsitektur set instruksi:

- arm64 - arsitektur ARM 64-bit, untuk AWS Prosesor Graviton.
- x86_64 - arsitektur x86 64-bit, untuk prosesor berbasis x86.

> **Note**
>
> Arsitektur arm64 tersedia di sebagian besar Wilayah AWS. Untuk informasi selengkapnya, silakan lihat AWS Lambda Harga. Dalam tabel harga memori, pilih Harga Arm tab, lalu buka Wilayah daftar dropdown untuk melihat yang mana Wilayah AWS mendukung arm64 dengan Lambda.
> Untuk contoh cara membuat fungsi dengan arsitektur arm64, lihat AWS Lambda Fungsi Didukung oleh AWS Prosesor Graviton2.

Topik

- Keuntungan menggunakan arsitektur arm64
- Persyaratan untuk migrasi ke arsitektur arm64
- Kompatibilitas kode fungsi dengan arsitektur arm64
- Cara bermigrasi ke arsitektur arm64
- Mengkonfigurasi arsitektur set instruksi

Keuntungan menggunakan arsitektur arm64

Fungsi lambda yang menggunakan arsitektur arm64 (AWS Prosesor Graviton) dapat mencapai harga dan kinerja yang jauh lebih baik daripada fungsi setara yang berjalan pada arsitektur x86_64. Pertimbangkan untuk menggunakan arm64 untuk aplikasi komputasi intensif seperti komputasi kinerja tinggi, pengkodean video, dan beban kerja simulasi.

Lambda mengelola perangkat keras yang mendasarinya dan tidak menjamin generasi prosesor tertentu: fungsi Anda mungkin berjalan pada generasi prosesor Graviton yang berbeda tergantung pada ketersediaan perangkat keras regional. Kode Anda tidak menargetkan chipset tertentu; itu

hanya perlu kompatibel dengan arsitektur arm64 (AArch64). Semua generasi Graviton memberikan manfaat ini dibandingkan x86_64:

- Cache L2 yang lebih besar per vCPU, yang mengurangi waktu baca memori dan meningkatkan latensi untuk backend web dan seluler, layanan mikro, dan sistem pemrosesan data.
- Peningkatan kinerja enkripsi melalui ekstensi kriptografi Arm.
- Dukungan untuk set instruksi yang menurunkan latensi CPU-based inferensi pembelajaran mesin.

Untuk informasi lebih lanjut, lihat AWS Prosesor Graviton.

Persyaratan untuk migrasi ke arsitektur arm64

Saat Anda memilih fungsi Lambda untuk bermigrasi ke arsitektur arm64, untuk memastikan migrasi yang lancar, pastikan fungsi Anda memenuhi persyaratan berikut:

- Paket penerapan hanya berisi komponen sumber terbuka dan kode sumber yang Anda kontrol, sehingga Anda dapat membuat pembaruan yang diperlukan untuk migrasi.
- Jika kode fungsi menyertakan dependensi pihak ketiga, setiap pustaka atau paket menyediakan versi arm64.

Kompatibilitas kode fungsi dengan arsitektur arm64

Kode fungsi Lambda Anda harus kompatibel dengan arsitektur set instruksi fungsi. Sebelum Anda memigrasikan fungsi ke arsitektur arm64, perhatikan poin-poin berikut tentang kode fungsi saat ini:

- Jika Anda menambahkan kode fungsi Anda menggunakan editor kode yang disematkan, kode Anda mungkin berjalan pada salah satu arsitektur tanpa modifikasi.
- Jika Anda mengunggah kode fungsi Anda, Anda harus mengunggah kode baru yang kompatibel dengan arsitektur target Anda.
- Jika fungsi Anda menggunakan lapisan, Anda harus periksa setiap lapisan untuk memastikan bahwa itu kompatibel dengan arsitektur baru. Jika layer tidak kompatibel, edit fungsi untuk mengganti versi layer saat ini dengan versi layer yang kompatibel.
- Jika fungsi Anda menggunakan ekstensi Lambda, Anda harus memeriksa setiap ekstensi untuk memastikan bahwa itu kompatibel dengan arsitektur baru.
- Jika fungsi Anda menggunakan jenis paket penyebaran gambar kontainer, Anda harus membuat image kontainer baru yang kompatibel dengan arsitektur fungsi.

Cara bermigrasi ke arsitektur arm64

Untuk memigrasikan fungsi Lambda ke arsitektur arm64, sebaiknya ikuti langkah-langkah berikut:

1. Buat daftar dependensi untuk aplikasi atau beban kerja Anda. Ketergantungan umum meliputi:
   - Semua pustaka dan paket yang digunakan fungsi.
   - Alat yang Anda gunakan untuk membangun, menerapkan, dan menguji fungsi, seperti kompiler, rangkaian pengujian, integrasi berkelanjutan, dan pengiriman berkelanjutan (CI/CD) pipeline, alat penyediaan, dan skrip.
   - Ekstensi Lambda dan alat pihak ketiga yang Anda gunakan untuk memantau fungsi dalam produksi.
2. Untuk setiap dependensi, periksa versi, lalu periksa apakah versi arm64 tersedia.
3. Bangun lingkungan untuk memigrasikan aplikasi Anda.
4. Bootstrap aplikasi.
5. Uji dan debug aplikasi.
6. Uji kinerja fungsi arm64. Bandingkan kinerja dengan versi x86_64.
7. Perbarui pipeline infrastruktur Anda untuk mendukung fungsi arm64 Lambda.
8. Tahap penerapan Anda ke produksi.

   Misalnya, gunakan konfigurasi routing alias untuk membagi lalu lintas antara versi x86 dan arm64 fungsi, dan membandingkan kinerja dan latensi.

Untuk informasi selengkapnya tentang cara membuat lingkungan kode untuk arsitektur arm64, termasuk informasi khusus bahasa untuk Java, Go, .NET, dan Python, lihat Memulai dengan AWS Graviton GitHub repositori.

Mengkonfigurasi arsitektur set instruksi

Anda dapat mengonfigurasi arsitektur set instruksi untuk fungsi Lambda baru dan yang ada menggunakan konsol Lambda, AWS SDK, AWS Command Line Interface (AWS CLI), atau CloudFormation. Ikuti langkah-langkah ini untuk mengubah arsitektur set instruksi untuk fungsi Lambda yang ada dari konsol.

1. Buka Halaman Fungsi Acara dari konsol Lambda.
2. Pilih nama fungsi yang ingin Anda konfigurasi arsitektur set instruksi.

3. Di bagian utama Kode tab, untuk Pengaturan runtime bagian, pilih Sunting.
4. Di bawah Arsitektur, pilih arsitektur set instruksi yang Anda ingin fungsi Anda gunakan.
5. Pilih Simpan.

Konfigurasikan batas waktu Fungsi Peristiwa Lambda

Lambda menjalankan kode Anda untuk jangka waktu tertentu sebelum waktu habis. Timeout adalah jumlah waktu maksimum dalam hitungan detik yang dapat dijalankan oleh fungsi Lambda. Nilai default untuk pengaturan ini adalah 3 detik, tetapi Anda dapat menyesuaikannya dengan penambahan 1 detik hingga nilai maksimum 900 detik (15 menit).

Untuk fungsi yang menggunakan Instans AWS Lambda Terkelola, pemanggilan asinkron dan pemanggilan pemetaan sumber peristiwa (kecuali Amazon MQ dan Amazon DocumentDB) mendukung nilai maksimum yang diizinkan 5.400 detik (90 menit).

Halaman ini menjelaskan bagaimana dan kapan harus memperbarui pengaturan batas waktu untuk fungsi Lambda.

Bagian-bagian

- Menentukan nilai batas waktu yang sesuai untuk fungsi Lambda
- Mengkonfigurasi batas waktu (konsol)
- Mengkonfigurasi batas waktu (AWS CLI)
- Mengkonfigurasi batas waktu (AWS SAM)

Menentukan nilai batas waktu yang sesuai untuk fungsi Lambda

Jika nilai batas waktu mendekati durasi rata-rata suatu fungsi, ada risiko yang lebih tinggi bahwa fungsi tersebut habis secara tak terduga. Durasi suatu fungsi dapat bervariasi berdasarkan jumlah transfer dan pemrosesan data, dan latensi layanan apa pun yang berinteraksi dengan fungsi tersebut. Beberapa penyebab umum batas waktu meliputi:

- Unduhan dari Amazon Simple Storage Service (Amazon S3) lebih besar atau memakan waktu lebih lama dari rata-rata.
- Fungsi membuat permintaan ke layanan lain, yang membutuhkan waktu lebih lama untuk merespons.
- Parameter yang disediakan untuk suatu fungsi membutuhkan lebih banyak kompleksitas komputasi dalam fungsi, yang menyebabkan pemanggilan memakan waktu lebih lama.

Saat menguji aplikasi Anda, pastikan pengujian Anda secara akurat mencerminkan ukuran dan kuantitas data serta nilai parameter yang realistis. Pengujian sering menggunakan sampel kecil

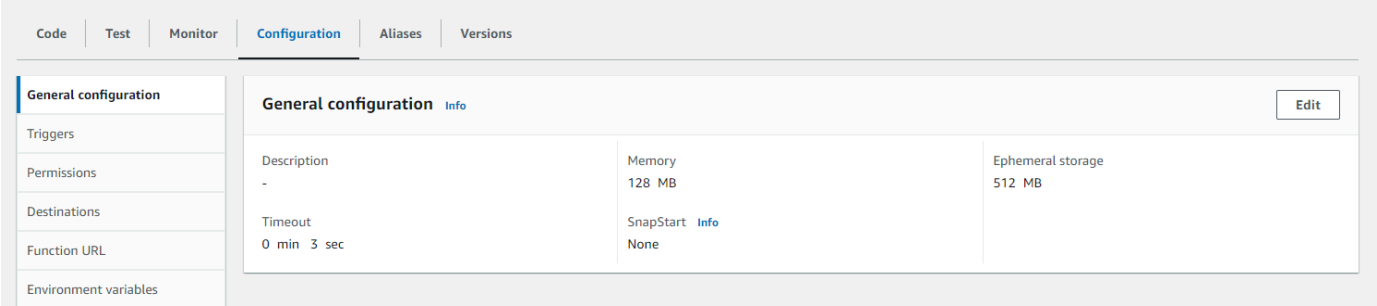
untuk kenyamanan, tetapi Anda harus menggunakan kumpulan data di batas atas dari apa yang diharapkan secara wajar untuk beban kerja Anda.

Mengkonfigurasi batas waktu (konsol)

Anda dapat mengonfigurasi batas waktu fungsi di konsol Lambda.

Untuk memodifikasi batas waktu untuk suatu fungsi

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih fungsi.
3. Pilih tab Konfigurasi dan kemudian pilih Konfigurasi umum.



4. Di bagian Konfigurasi umum, pilih Edit.
5. Untuk Timeout, tetapkan nilai antara 1 dan 900 detik (15 menit).
6. Pilih Simpan.

Mengkonfigurasi batas waktu (AWS CLI)

Anda dapat menggunakan perintah update-function-configuration untuk mengonfigurasi nilai batas waktu, dalam hitungan detik. Contoh perintah berikut meningkatkan batas waktu fungsi menjadi 120 detik (2 menit).

Example

```
aws lambda update-function-configuration \
  --function-name my-function \
  --timeout 120
```

Mengkonfigurasi batas waktu (AWS SAM)

Anda dapat menggunakan AWS Serverless Application Modeluntuk mengkonfigurasi nilai batas waktu untuk fungsi Anda. Perbarui properti Timeout di `template.yaml` file Anda dan kemudian jalankan sam deploy.

Example template.yaml

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: An AWS Serverless Application Model template describing your function.
Resources:
  my-function:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: .
      Description: ''
      MemorySize: 128
      Timeout: 120
      # Other function properties...
```

Bekerja dengan variabel lingkungan Lambda

Anda dapat menggunakan variabel lingkungan untuk menyesuaikan perilaku fungsi Anda tanpa memperbarui kode. Variabel lingkungan adalah sepasang string yang disimpan dalam konfigurasi spesifik versi fungsi. Runtime Lambda membuat variabel lingkungan tersedia bagi kode Anda dan mengatur variabel lingkungan tambahan yang berisi informasi tentang fungsi dan permintaan invokasi.

> **Note**
>
> Untuk meningkatkan keamanan, sebaiknya Anda menggunakan AWS Secrets Manager variabel lingkungan untuk menyimpan kredensi database dan informasi sensitif lainnya seperti kunci API atau token otorisasi. Untuk informasi selengkapnya, lihat Gunakan rahasia Secrets Manager di Lambda Event Functions.

Variabel lingkungan tidak dievaluasi sebelum pemanggilan fungsi. Nilai apa pun yang Anda tentukan dianggap sebagai string harafiah dan tidak diperluas. Lakukan evaluasi variabel dalam kode fungsi Anda.

Membuat variabel lingkungan Lambda

Anda dapat mengonfigurasi variabel lingkungan di Lambda menggunakan konsol Lambda, (), AWS Command Line Interface AWS Serverless Application Model (AWS CLIAWS SAM), atau menggunakan SDK. AWS

Console

Anda menentukan variabel lingkungan pada versi fungsi yang belum dipublikasikan. Saat Anda mempublikasikan versi, variabel lingkungan dikunci untuk versi tersebut bersama dengan pengaturan konfigurasi khusus versi lainnya.

Anda membuat variabel lingkungan untuk fungsi Anda dengan mendefinisikan kunci dan nilai. Fungsi Anda menggunakan nama kunci untuk mengambil nilai variabel lingkungan.

Untuk mengatur variabel lingkungan di konsol Lambda

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih fungsi.

3. Pilih tab Konfigurasi, lalu pilih variabel Lingkungan.
4. Pada Variabel lingkungan, pilih Edit.
5. Pilih Tambahkan variabel lingkungan.
6. Masukkan kunci dan nilai.

   Persyaratan

   - Kunci dimulai dengan huruf dan setidaknya dua karakter.
   - Kunci hanya berisi huruf, angka, dan karakter garis bawah (_).
   - Kunci tidak dicadangkan oleh Lambda.
   - Ukuran total semua variabel lingkungan tidak lebih dari 4 KB.
7. Pilih Simpan.

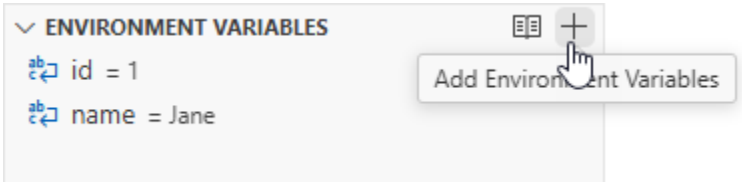
Untuk menghasilkan daftar variabel lingkungan di editor kode konsol

Anda dapat membuat daftar variabel lingkungan di editor kode Lambda. Ini adalah cara cepat untuk mereferensikan variabel lingkungan Anda saat Anda membuat kode.

1. Pilih tab Kode.
2. Gulir ke bawah ke bagian VARIABEL LINGKUNGAN dari editor kode. Variabel lingkungan yang ada tercantum di sini:



3. Untuk membuat variabel lingkungan baru, pilih pilih tanda plus (+):



Variabel lingkungan tetap dienkripsi saat terdaftar di editor kode konsol. Jika Anda mengaktifkan pembantu enkripsi untuk enkripsi dalam perjalanan, maka pengaturan tersebut tetap tidak berubah. Untuk informasi selengkapnya, lihat Mengamankan variabel lingkungan Lambda.

Daftar variabel lingkungan hanya-baca dan hanya tersedia di konsol Lambda. File ini tidak disertakan saat Anda mengunduh arsip file.zip fungsi, dan Anda tidak dapat menambahkan variabel lingkungan dengan mengunggah file ini.

AWS CLI

Contoh berikut menetapkan dua variabel lingkungan pada fungsi yang diberi nama `my-function`.

```
aws lambda update-function-configuration \
  --function-name my-function \
  --environment "Variables={BUCKET=amzn-s3-demo-bucket,KEY=file.txt}"
```

Ketika Anda menerapkan variabel lingkungan dengan perintah `update-function-configuration`, seluruh isi dari struktur `Variables` digantikan. Untuk mempertahankan variabel lingkungan yang ada saat Anda menambahkan yang baru, sertakan semua nilai yang ada dalam permintaan Anda.

Untuk mendapatkan konfigurasi saat ini, gunakan perintah `get-function-configuration`.

```
aws lambda get-function-configuration \
  --function-name my-function
```

Anda akan melihat output berikut:

```
{
    "FunctionName": "my-function",
    "FunctionArn": "arn:aws:lambda:us-east-2:111122223333:function:my-function",
    "Runtime": "nodejs24.x",
    "Role": "arn:aws:iam::111122223333:role/lambda-role",
    "Environment": {
        "Variables": {
            "BUCKET": "amzn-s3-demo-bucket",
            "KEY": "file.txt"
        }
    },
    "RevisionId": "0894d3c1-2a3d-4d48-bf7f-abade99f3c15",
    ...
}
```

Anda dapat meneruskan ID revisi dari output `get-function-configuration` sebagai parameter ke`update-function-configuration`. Ini memastikan bahwa nilai tidak berubah antara saat Anda membaca konfigurasi dan saat Anda memperbaruinya.

Untuk mengonfigurasi kunci enkripsi fungsi, atur opsi `KMSKeyARN`.

```
aws lambda update-function-configuration \
  --function-name my-function \
  --kms-key-arn arn:aws:kms:us-east-2:111122223333:key/055efbb4-xmpl-4336-
ba9c-538c7d31f599
```

AWS SAM

Anda dapat menggunakan AWS Serverless Application Modeluntuk mengkonfigurasi variabel lingkungan untuk fungsi Anda. Perbarui properti Lingkungan dan Variabel dalam `template.yaml` file Anda dan kemudian jalankan sam deploy.

Example template.yaml

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: An AWS Serverless Application Model template describing your function.
Resources:
  my-function:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: .
      Description: ''
      MemorySize: 128
      Timeout: 120
      Handler: index.handler
      Runtime: nodejs24.x
      Architectures:
        - x86_64
      EphemeralStorage:
        Size: 10240
      Environment:
        Variables:
          BUCKET: amzn-s3-demo-bucket
          KEY: file.txt
      # Other function properties...
```

AWS SDKs

Untuk mengelola variabel lingkungan menggunakan AWS SDK, gunakan operasi API berikut.

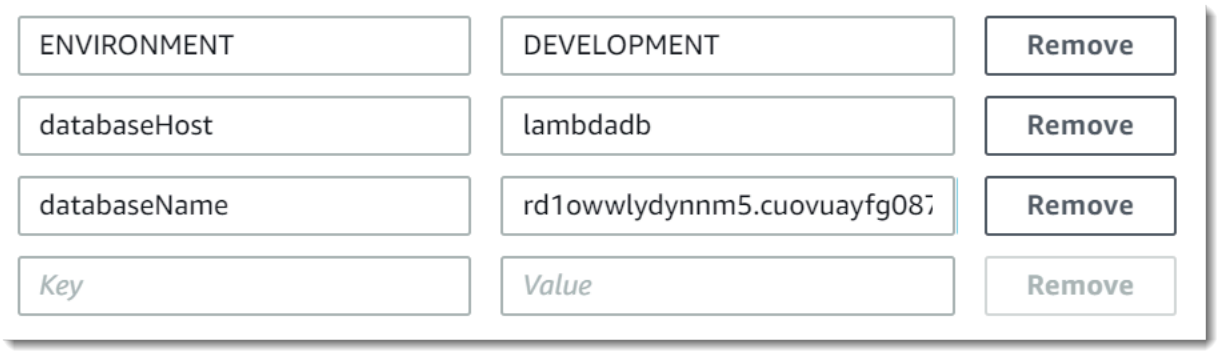
- UpdateFunctionConfiguration
- GetFunctionConfiguration
- CreateFunction

Untuk mempelajari lebih lanjut, lihat dokumentasi AWS SDK untuk bahasa pemrograman pilihan Anda.

Contoh skenario untuk variabel lingkungan

Anda dapat menggunakan variabel lingkungan untuk menyesuaikan perilaku fungsi di lingkungan pengujian Anda dan lingkungan produksi. Misalnya, Anda dapat membuat dua fungsi dengan kode yang sama, tetapi konfigurasi yang berbeda. Satu fungsi terhubung ke basis data uji, dan fungsi lainnya terhubung ke basis data produksi. Dalam situasi ini, Anda menggunakan variabel lingkungan untuk meneruskan nama host dan detail koneksi lainnya untuk database ke fungsi.

Contoh berikut menunjukkan cara menentukan host basis data dan nama basis data sebagai variabel lingkungan.



Jika Anda ingin lingkungan pengujian Anda menghasilkan lebih banyak informasi debug daripada lingkungan produksi, Anda dapat mengatur variabel lingkungan untuk mengonfigurasi lingkungan pengujian Anda untuk menggunakan lebih banyak log verbose atau pelacakan yang lebih terperinci.

Misalnya, di lingkungan pengujian, Anda dapat menyetel variabel lingkungan dengan kunci `LOG_LEVEL` dan nilai yang menunjukkan tingkat log debug atau jejak. Dalam kode fungsi Lambda Anda, Anda kemudian dapat menggunakan variabel lingkungan ini untuk mengatur tingkat log.

Berikut contoh kode di Python dan Node.js menggambarkan bagaimana Anda dapat mencapai ini. Contoh-contoh ini mengasumsikan variabel lingkungan Anda memiliki nilai `DEBUG` dalam Python atau `debug` in. Node.js

Python

Example Kode Python untuk mengatur level log

```
import os
import logging

# Initialize the logger
logger = logging.getLogger()

# Get the log level from the environment variable and default to INFO if not set
log_level = os.environ.get('LOG_LEVEL', 'INFO')

# Set the log level
logger.setLevel(log_level)

def lambda_handler(event, context):
    # Produce some example log outputs
    logger.debug('This is a log with detailed debug information - shown only in test
 environment')
    logger.info('This is a log with standard information - shown in production and
 test environments')
```

Node.js (ES module format)

Example Node.js kode untuk mengatur tingkat log

Contoh ini menggunakan library `winston` logging. Gunakan npm untuk menambahkan pustaka ini ke paket penerapan fungsi Anda. Untuk informasi selengkapnya, lihat the section called “Membuat paket penerapan.zip dengan dependensi”.

```
import winston from 'winston';

// Initialize the logger using the log level from environment variables, defaulting
 to INFO if not set
const logger = winston.createLogger({
```

```
   level: process.env.LOG_LEVEL || 'info',
   format: winston.format.json(),
   transports: [new winston.transports.Console()]
});

export const handler = async (event) => {
   // Produce some example log outputs
   logger.debug('This is a log with detailed debug information - shown only in test
 environment');
   logger.info('This is a log with standard information - shown in production and
 test environment');

};
```

Mengambil variabel lingkungan Lambda

Untuk mengambil variabel lingkungan dalam kode fungsi Anda, gunakan metode standar untuk bahasa pemrograman Anda.

Node.js

```
let region = process.env.AWS_REGION
```

Python

```
import os
  region = os.environ['AWS_REGION']
```

> **Note**
>
> Dalam beberapa kasus, Anda mungkin perlu menggunakan format berikut:
>
> ```
> region = os.environ.get('AWS_REGION')
> ```

Ruby

```
region = ENV["AWS_REGION"]
```

Java

```
String region = System.getenv("AWS_REGION");
```

Go

```
var region = os.Getenv("AWS_REGION")
```

C#

```
string region = Environment.GetEnvironmentVariable("AWS_REGION");
```

PowerShell

```
$region = $env:AWS_REGION
```

Lambda menyimpan variabel lingkungan secara aman dengan mengenkripsinya saat istirahat. Anda dapat mengonfigurasi Lambda untuk menggunakan kunci enkripsi yang berbeda, mengenkripsi nilai variabel lingkungan di sisi klien, atau mengatur variabel lingkungan CloudFormation dalam templat dengan. AWS Secrets Manager

Variabel lingkungan runtime yang ditetapkan

Runtime Lambda menetapkan beberapa variabel lingkungan selama inisialisasi. Sebagian besar variabel lingkungan memberikan informasi tentang fungsi atau runtime. Kunci untuk variabel lingkungan ini dicadangkan dan tidak dapat diatur dalam konfigurasi fungsi Anda.

Variabel lingkungan yang tersimpan

- `_HANDLER` – Lokasi handler dikonfigurasi pada fungsi.
- `_X_AMZN_TRACE_ID` – Header pelacakan X-Ray . Variabel lingkungan ini berubah dengan setiap pemanggilan.
  - Variabel lingkungan ini tidak ditentukan untuk OS-only runtime (keluarga `provided` runtime). Anda dapat mengatur `_X_AMZN_TRACE_ID` runtime kustom menggunakan header `Lambda-Runtime-Trace-Id` respons dari file. Invokasi berikutnya
  - Untuk Java runtime versi 17 dan yang lebih baru, variabel lingkungan ini tidak digunakan. Sebaliknya, Lambda menyimpan informasi penelusuran di properti sistem. `com.amazonaws.xray.traceHeader`

- `AWS_DEFAULT_REGION`— Default Wilayah AWS di mana fungsi Lambda dijalankan.
- `AWS_REGION`— Wilayah AWS Tempat fungsi Lambda dijalankan. Jika didefinisikan, nilai ini mengesampingkan. `AWS_DEFAULT_REGION`
  - Untuk informasi selengkapnya tentang penggunaan variabel Wilayah AWS lingkungan dengan AWS SDK, lihat AWS Wilayah dalam Panduan Referensi AWS SDK dan Alat.
- `AWS_EXECUTION_ENV`— Pengidentifikasi runtime, diawali oleh `AWS_Lambda_` (misalnya,). `AWS_Lambda_java8` Variabel lingkungan ini tidak ditentukan untuk OS-only runtime (keluarga `provided` runtime).
- `AWS_LAMBDA_FUNCTION_NAME` – Nama fungsi.
- `AWS_LAMBDA_FUNCTION_MEMORY_SIZE` – Jumlah memori yang tersedia untuk fungsi dalam MB.
- `AWS_LAMBDA_FUNCTION_VERSION` – Versi fungsi yang sedang dijalankan.
- `AWS_LAMBDA_INITIALIZATION_TYPE`— Jenis inisialisasi fungsi, yaitu,`on-demand`, `provisioned-concurrency`snap-start, ataulambda-managed-instances. Untuk selengkapnya, lihat Mengonfigurasi konkurensi yang disediakan,, atau. Meningkatkan kinerja startup dengan Lambda SnapStart Instans Terkelola Lambda
- `AWS_LAMBDA_LOG_GROUP_NAME`, `AWS_LAMBDA_LOG_STREAM_NAME` — Nama grup Amazon CloudWatch Logs dan streaming untuk fungsi tersebut. Variabel `AWS_LAMBDA_LOG_GROUP_NAME` dan `AWS_LAMBDA_LOG_STREAM_NAME` lingkungan tidak tersedia dalam fungsi Lambda SnapStart .
- `AWS_ACCESS_KEY`,`AWS_ACCESS_KEY_ID`,`AWS_SECRET_ACCESS_KEY`, `AWS_SESSION_TOKEN` — Kunci akses yang diperoleh dari peran eksekusi fungsi.
- `AWS_LAMBDA_RUNTIME_API` – (Runtime kustom) Host dan port API runtime.
- `LAMBDA_TASK_ROOT` – Jalur ke kode fungsi Lambda Anda.
- `LAMBDA_RUNTIME_DIR` – Jalur ke pustaka runtime.
- `AWS_LAMBDA_MAX_CONCURRENCY`— (Hanya Instans Terkelola Lambda) Jumlah maksimum pemanggilan bersamaan yang akan dikirim Lambda ke satu lingkungan eksekusi.
- `AWS_LAMBDA_METADATA_API`— Alamat server titik akhir metadata dalam format `{ipv4_address}:{port}` (misalnya,). `169.254.100.1:9001`
- `AWS_LAMBDA_METADATA_TOKEN`— Token otentikasi unik untuk lingkungan eksekusi saat ini yang digunakan untuk mengautentikasi permintaan ke titik akhir metadata. Lambda menghasilkan token ini secara otomatis saat inisialisasi.

Variabel lingkungan tambahan berikut ini tidak dicadangkan dan dapat diperluas dalam konfigurasi fungsi Anda.

Variabel lingkungan yang tidak dicadangkan

- `LANG` – Lokal runtime (`en_US.UTF-8`).
- `PATH` – Jalur eksekusi (`/usr/local/bin:/usr/bin/:/bin:/opt/bin`).
- `LD_LIBRARY_PATH` – Jalur pustaka sistem (`/var/lang/lib:/lib64:/usr/lib64:$LAMBDA_RUNTIME_DIR:$LAMBDA_RUNTIME_DIR/lib:$LAMBDA_TASK_ROOT:$LAMBDA_TASK_ROOT/lib:/opt/lib`).
- `NODE_PATH`— (Node.js) Jalur Node.js perpustakaan (`/opt/nodejs/node12/node_modules/:/opt/nodejs/node_modules:$LAMBDA_RUNTIME_DIR/node_modules`).
- `NODE_OPTIONS`— (Node.js) Untuk Node.js runtime, Anda dapat menggunakan `NODE_OPTIONS` untuk mengaktifkan kembali fitur eksperimental yang dinonaktifkan Lambda secara default.
- `PYTHONPATH`— (Python) Jalur perpustakaan Python (). `$LAMBDA_RUNTIME_DIR`
- `GEM_PATH` – (Ruby) Jalur pustaka Ruby (`$LAMBDA_TASK_ROOT/vendor/bundle/ruby/3.3.0:/opt/ruby/gems/3.3.0`).
- `AWS_XRAY_CONTEXT_MISSING`— Untuk X-Ray melacak, Lambda menyetelnya `LOG_ERROR` untuk menghindari kesalahan runtime dari SDK. X-Ray
- `AWS_XRAY_DAEMON_ADDRESS`— Untuk X-Ray melacak, alamat IP dan port X-Ray daemon.
- `AWS_LAMBDA_DOTNET_PREJIT`— (.NET) Atur variabel ini untuk mengaktifkan atau menonaktifkan optimasi runtime tertentu.NET. Nilai mencakup `always`, `never`, dan `provisioned-concurrency`. Untuk informasi selengkapnya, lihat Mengkonfigurasi konkurensi yang disediakan untuk Fungsi Peristiwa.
- `TZ` – Zona waktu lingkungan (`:UTC`). Lingkungan eksekusi menggunakan NTP untuk menyinkronkan jam sistem.

Nilai sampel yang ditampilkan mencerminkan runtime terbaru. Adanya variabel tertentu atau nilainya dapat bervariasi pada runtime awal.

Mengamankan variabel lingkungan Lambda

Untuk mengamankan variabel lingkungan Anda, Anda dapat menggunakan enkripsi sisi server untuk melindungi data Anda saat istirahat dan enkripsi sisi klien untuk melindungi data Anda dalam perjalanan.

> **Note**
>
> Untuk meningkatkan keamanan database, kami sarankan Anda menggunakan AWS Secrets Manager bukan variabel lingkungan untuk menyimpan kredensi database. Untuk informasi selengkapnya, lihat Gunakan rahasia Secrets Manager di Lambda Event Functions.

Keamanan saat istirahat

Lambda selalu menyediakan enkripsi sisi server saat istirahat dengan file. AWS KMS key Secara default, Lambda menggunakan file. Kunci yang dikelola AWS Jika perilaku default ini sesuai dengan alur kerja Anda, Anda tidak perlu menyiapkan hal lain. Lambda membuat Kunci yang dikelola AWS di akun Anda dan mengelola izin untuk Anda. AWS tidak membebankan biaya untuk menggunakan kunci ini.

Jika mau, Anda dapat memberikan kunci yang dikelola AWS KMS pelanggan sebagai gantinya. Anda dapat melakukan ini untuk memiliki kontrol atas rotasi kunci KMS atau untuk memenuhi persyaratan organisasi Anda untuk mengelola kunci KMS. Saat Anda menggunakan kunci yang dikelola pelanggan, hanya pengguna di akun Anda dengan akses ke kunci KMS yang dapat melihat atau mengelola variabel lingkungan pada fungsi tersebut.

Kunci yang dikelola pelanggan dikenakan AWS KMS biaya standar. Untuk informasi selengkapnya, lihat harga AWS Key Management Service.

Keamanan dalam perjalanan

Untuk keamanan tambahan, Anda dapat mengaktifkan helper untuk enkripsi dalam perjalanan, yang memastikan bahwa variabel lingkungan Anda dienkripsi sisi klien untuk perlindungan dalam perjalanan.

Untuk mengonfigurasi enkripsi untuk variabel lingkungan Anda

1. Gunakan AWS Key Management Service (AWS KMS) untuk membuat kunci terkelola pelanggan apa pun yang digunakan Lambda untuk enkripsi sisi server dan sisi klien. Untuk informasi selengkapnya, lihat Membuat kunci di Panduan AWS Key Management Service Pengembang.
2. Menggunakan konsol Lambda, navigasikan ke halaman Edit variabel lingkungan.
    a. Buka halaman Fungsi Acara konsol Lambda.
    b. Pilih fungsi.

c. Pilih Konfigurasi, lalu pilih variabel Lingkungan dari bilah navigasi kiri.

d. Di bagian Variabel lingkungan, pilih Edit.

e. Perluas Konfigurasi enkripsi.

3. (Opsional) Aktifkan pembantu enkripsi konsol untuk menggunakan enkripsi sisi klien untuk melindungi data Anda saat transit.

   a. Di bawah Enkripsi dalam perjalanan, pilih Aktifkan pembantu untuk enkripsi saat transit.

   b. Untuk setiap variabel lingkungan yang ingin Anda aktifkan pembantu enkripsi konsol, pilih Enkripsi di sebelah variabel lingkungan.

   c. Di bawah AWS KMS key untuk mengenkripsi saat transit, pilih kunci terkelola pelanggan yang Anda buat di awal prosedur ini.

   d. Pilih Kebijakan peran eksekusi dan salin kebijakan. Kebijakan ini memberikan izin untuk peran eksekusi fungsi Anda untuk mendekripsi variabel lingkungan.

      Simpan kebijakan ini untuk digunakan pada langkah terakhir prosedur ini.

   e. Tambahkan kode ke fungsi Anda yang mendekripsi variabel lingkungan. Untuk melihat contoh, pilih Dekripsi cuplikan rahasia.

4. (Opsional) Tentukan kunci terkelola pelanggan Anda untuk enkripsi saat istirahat.

   a. Pilih Gunakan kunci utama pelanggan.

   b. Pilih kunci yang dikelola pelanggan yang Anda buat di awal prosedur ini.

5. Pilih Simpan.

6. Menyiapkan izin.

   Jika Anda menggunakan kunci yang dikelola pelanggan dengan enkripsi sisi server, berikan izin kepada pengguna atau peran apa pun yang Anda inginkan untuk dapat melihat atau mengelola variabel lingkungan pada fungsi tersebut. Untuk informasi selengkapnya, lihat Mengelola izin ke kunci KMS enkripsi sisi server Anda.

   Jika Anda mengaktifkan enkripsi sisi klien untuk keamanan dalam perjalanan, fungsi Anda memerlukan izin untuk memanggil operasi API. `kms:Decrypt` Tambahkan kebijakan yang Anda simpan sebelumnya dalam prosedur ini ke peran eksekusi fungsi.

Mengelola izin ke kunci KMS enkripsi sisi server Anda

Tidak ada AWS KMS izin yang diperlukan untuk pengguna Anda atau peran eksekusi fungsi untuk menggunakan kunci enkripsi default. Untuk menggunakan kunci yang dikelola pelanggan, Anda memerlukan izin untuk menggunakan kunci tersebut. Lambda menggunakan izin Anda untuk membuat izin pada kunci. Ini memungkinkan Lambda menggunakannya untuk enkripsi.

- `kms:ListAliases` – Untuk melihat kunci di konsol Lambda.
- `kms:CreateGrant`, `kms:Encrypt` — Untuk mengonfigurasi kunci yang dikelola pelanggan pada suatu fungsi.
- `kms:Decrypt`— Untuk melihat dan mengelola variabel lingkungan yang dienkripsi dengan kunci yang dikelola pelanggan.

Anda bisa mendapatkan izin ini dari Akun AWS atau dari kebijakan izin berbasis sumber daya kunci. `ListAliases`disediakan oleh kebijakan terkelola untuk Lambda. Kebijakan kunci memberikan izin yang tersisa kepada pengguna di grup Pengguna utama.

Pengguna tanpa `Decrypt` izin masih dapat mengelola fungsi, tetapi mereka tidak dapat melihat variabel lingkungan atau mengelolanya di konsol Lambda. Untuk mencegah pengguna melihat variabel lingkungan, tambahkan pernyataan ke izin pengguna yang menolak akses ke kunci default, kunci yang dikelola pelanggan, atau semua kunci.

Example Kebijakan IAM – Menolak akses dengan ARN kunci

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Deny",
            "Action": [
                "kms:Decrypt"
            ],
            "Resource": "arn:aws:kms:us-east-2:111122223333:key/3be10e2d-
xmpl-4be4-bc9d-0405a71945cc"
        }
    ]
```

```
}
```

Untuk detail tentang mengelola izin utama, lihat Kebijakan utama AWS KMS di Panduan AWS Key Management Service Pengembang.

Memberikan Lambda Event Functions akses ke sumber daya di Amazon VPC

Dengan Amazon Virtual Private Cloud (Amazon VPC), Anda dapat membuat jaringan pribadi di sumber daya host Anda seperti Akun AWS instans Amazon Elastic Compute Cloud (Amazon EC2), instans Amazon Relational Database Service (Amazon RDS), dan instans Amazon. ElastiCache Anda dapat memberikan akses fungsi Lambda ke sumber daya yang dihosting di VPC Amazon dengan melampirkan fungsi Anda ke VPC melalui subnet pribadi yang berisi sumber daya. Ikuti petunjuk di bagian berikut untuk melampirkan fungsi Lambda ke VPC Amazon menggunakan konsol Lambda, (), atau. AWS Command Line Interface AWS CLI AWS SAM

> **Note**
>
> Setiap fungsi Lambda berjalan di dalam VPC yang dimiliki dan dikelola oleh layanan Lambda. VPC ini dikelola secara otomatis oleh Lambda dan tidak terlihat oleh pelanggan. Mengonfigurasi fungsi Anda untuk mengakses AWS sumber daya lain di VPC Amazon tidak berpengaruh pada Lambda-managed VPC yang dijalankan fungsi Anda di dalamnya.

Bagian-bagian

- Izin IAM yang diperlukan
- Melampirkan fungsi Lambda ke VPC Amazon di Akun AWS
- Akses internet saat dilampirkan ke VPC
- Dukungan IPv6
- Praktik terbaik untuk menggunakan Lambda dengan Amazon VPC
- Memahami Antarmuka Jaringan Elastis Hyperplane (ENI)
- Menggunakan kunci syarat IAM untuk pengaturan VPC
- Tutorial VPC

Izin IAM yang diperlukan

Untuk melampirkan fungsi Lambda ke VPC Amazon di Anda, Akun AWS Lambda memerlukan izin untuk membuat dan mengelola antarmuka jaringan yang digunakannya untuk memberikan akses fungsi Anda ke sumber daya di VPC.

Antarmuka jaringan yang dibuat Lambda dikenal sebagai Hyperplane Elastic Network Interfaces, atau Hyperplane ENIs. Untuk mempelajari lebih lanjut tentang antarmuka jaringan ini, lihatthe section called “Memahami Antarmuka Jaringan Elastis Hyperplane (ENI)”.

Anda dapat memberikan fungsi izin yang dibutuhkan dengan melampirkan kebijakan AWS terkelola AWSLambdaVPCAccessExecutionRoleke peran eksekusi fungsi Anda. Saat Anda membuat fungsi baru di konsol Lambda dan melampirkannya ke VPC, Lambda secara otomatis menambahkan kebijakan izin ini untuk Anda.

Jika Anda lebih suka membuat kebijakan izin IAM Anda sendiri, pastikan untuk menambahkan semua izin berikut dan mengizinkannya di semua sumber daya (): `"Resource": "*"`

- EC2: CreateNetworkInterface
- EC2: DescribeNetworkInterfaces
- EC2: DescribeSubnets
- EC2: DeleteNetworkInterface
- EC2: AssignPrivateIpAddresses
- EC2: UnassignPrivateIpAddresses

Perhatikan bahwa peran fungsi Anda hanya memerlukan izin ini untuk membuat antarmuka jaringan, bukan untuk memanggil fungsi Anda. Anda masih dapat menjalankan fungsi dengan sukses saat dilampirkan ke VPC Amazon, meskipun Anda menghapus izin ini dari peran eksekusi fungsi Anda.

Untuk melampirkan fungsi Anda ke VPC, Lambda juga perlu memverifikasi sumber daya jaringan menggunakan peran pengguna IAM Anda. Pastikan peran pengguna Anda memiliki izin IAM berikut:

- EC2: DescribeSecurityGroups
- EC2: DescribeSubnets
- EC2: DescribeVpcs
- EC2: GetSecurityGroupsForVpc

> **Note**
>
> Izin Amazon EC2 yang Anda berikan ke peran eksekusi fungsi Anda digunakan oleh layanan Lambda untuk melampirkan fungsi Anda ke VPC. Namun, Anda juga secara implisit memberikan izin ini ke kode fungsi Anda. Ini berarti bahwa kode fungsi Anda dapat membuat

panggilan API Amazon EC2 ini. Untuk saran tentang mengikuti praktik terbaik keamanan, lihatthe section called “Praktik terbaik keamanan”.

Melampirkan fungsi Lambda ke VPC Amazon di Akun AWS

Lampirkan fungsi Anda ke VPC Amazon di Anda Akun AWS dengan menggunakan konsol Lambda, atau. AWS CLI AWS SAM Jika Anda menggunakan AWS CLI atau AWS SAM, atau melampirkan fungsi yang ada ke VPC menggunakan konsol Lambda, pastikan bahwa peran eksekusi fungsi Anda memiliki izin yang diperlukan yang tercantum di bagian sebelumnya.

Fungsi Lambda tidak dapat terhubung langsung ke VPC dengan tenancy instans khusus. Untuk terhubung ke sumber daya dalam VPC khusus, sambungkan ke VPC kedua dengan penyewaan default.

Lambda console

Untuk melampirkan fungsi ke VPC Amazon saat Anda membuatnya

1. Buka halaman Fungsi Acara konsol Lambda dan pilih Buat Fungsi Acara.
2. Di bawah Informasi Dasar, untuk Nama fungsi, masukkan nama untuk fungsi Anda.
3. Konfigurasikan pengaturan VPC untuk fungsi dengan melakukan hal berikut:
   a. Perluas Pengaturan lanjutan.
   b. Pilih Aktifkan VPC, lalu pilih VPC yang ingin Anda lampirkan fungsinya.
   c. (Opsional) Untuk mengizinkan lalu lintas IPv6 keluar, pilih Izinkan lalu lintas IPv6 untuk subnet dual-stack.
   d. Pilih subnet dan grup keamanan untuk membuat antarmuka jaringan. Jika Anda memilih Izinkan lalu lintas IPv6 untuk subnet dual-stack, semua subnet yang dipilih harus memiliki blok CIDR IPv4 dan blok CIDR IPv6.

> **Note**
>
> Untuk mengakses sumber daya privat, sambungkan fungsi Anda ke subnet privat. Jika fungsi Anda membutuhkan akses internet, lihatthe section called “Akses internet untuk fungsi VPC”. Menghubungkan fungsi ke subnet publik tidak memberikan akses internet atau alamat IP publik.

4. Pilih Buat Fungsi Acara.

Untuk melampirkan fungsi yang ada ke VPC Amazon

1. Buka halaman Fungsi Acara konsol Lambda dan pilih fungsi Anda.
2. Pilih tab Konfigurasi, lalu pilih VPC.
3. Pilih Edit.
4. Di bawah VPC, pilih VPC Amazon yang ingin Anda lampirkan fungsinya.
5. (Opsional) Untuk mengizinkan lalu lintas IPv6 keluar, pilih Izinkan lalu lintas IPv6 untuk subnet dual-stack.
6. Pilih subnet dan grup keamanan untuk membuat antarmuka jaringan. Jika Anda memilih Izinkan lalu lintas IPv6 untuk subnet dual-stack, semua subnet yang dipilih harus memiliki blok CIDR IPv4 dan blok CIDR IPv6.

> **Note**
>
> Untuk mengakses sumber daya privat, sambungkan fungsi Anda ke subnet privat. Jika fungsi Anda membutuhkan akses internet, lihatthe section called “Akses internet untuk fungsi VPC”. Menghubungkan fungsi ke subnet publik tidak memberikan akses internet atau alamat IP publik.

7. Pilih Simpan.

AWS CLI

Untuk melampirkan fungsi ke VPC Amazon saat Anda membuatnya

- Untuk membuat fungsi Lambda dan melampirkannya ke VPC, jalankan perintah CLI berikut. `create-function`

```
aws lambda create-function --function-name my-function \
--runtime nodejs24.x --handler index.js --zip-file fileb://function.zip \
--role arn:aws:iam::123456789012:role/lambda-role \
--vpc-config
 Ipv6AllowedForDualStack=true,SubnetIds=subnet-071f712345678e7c8,subnet-07fd123456788a03
```

Tentukan subnet dan grup keamanan Anda sendiri dan atur `Ipv6AllowedForDualStack` ke `true` atau `false` sesuai dengan kasus penggunaan Anda.

Untuk melampirkan fungsi yang ada ke VPC Amazon

- Untuk melampirkan fungsi yang ada ke VPC, jalankan perintah CLI berikut. `update-function-configuration`

```
aws lambda update-function-configuration --function-name my-function \
--vpc-config Ipv6AllowedForDualStack=true,
 SubnetIds=subnet-071f712345678e7c8,subnet-07fd123456788a036,SecurityGroupIds=sg-0859123
```

Untuk melepaskan fungsi Anda dari VPC

- Untuk melepaskan fungsi Anda dari VPC, jalankan perintah `update-function-configuration` CLI berikut dengan daftar kosong subnet VPC dan grup keamanan.

```
aws lambda update-function-configuration --function-name my-function \
--vpc-config SubnetIds=[],SecurityGroupIds=[]
```

AWS SAM

Untuk melampirkan fungsi Anda ke VPC

- Untuk melampirkan fungsi Lambda ke VPC Amazon, tambahkan `VpcConfig` properti ke definisi fungsi Anda seperti yang ditunjukkan pada contoh template berikut. Untuk informasi selengkapnya tentang properti ini, lihat AWS: :Lambda: :Function VpcConfig di Panduan CloudFormation Pengguna (AWS SAM `VpcConfig`properti diteruskan langsung ke `VpcConfig` properti sumber daya). CloudFormation `AWS::Lambda::Function`

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31

Resources:
  MyFunction:
    Type: AWS::Serverless::Function
    Properties:
```

```
      CodeUri: ./lambda_function/
      Handler: lambda_function.handler
      Runtime: python3.12
      VpcConfig:
        SecurityGroupIds:
          - !Ref MySecurityGroup
        SubnetIds:
          - !Ref MySubnet1
          - !Ref MySubnet2
      Policies:
        - AWSLambdaVPCAccessExecutionRole

  MySecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: Security group for Lambda function
      VpcId: !Ref MyVPC

  MySubnet1:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref MyVPC
      CidrBlock: 10.0.1.0/24

  MySubnet2:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref MyVPC
      CidrBlock: 10.0.2.0/24

  MyVPC:
    Type: AWS::EC2::VPC
    Properties:
      CidrBlock: 10.0.0.0/16
```

Untuk informasi selengkapnya tentang mengonfigurasi VPC Anda, AWS lihat: :EC2: :VPC AWS SAM di Panduan Pengguna.CloudFormation

Akses internet saat dilampirkan ke VPC

Secara default, fungsi Lambda memiliki akses ke internet publik. Ketika Anda melampirkan fungsi Anda ke VPC, itu hanya dapat mengakses sumber daya yang tersedia dalam VPC itu. Untuk

memberikan akses fungsi Anda ke internet, Anda juga perlu mengkonfigurasi VPC untuk memiliki akses internet. Untuk mempelajari selengkapnya, lihat the section called “Akses internet untuk fungsi VPC”.

Dukungan IPv6

Fungsi Anda dapat terhubung ke sumber daya dalam subnet VPC dual-stack melalui IPv6. Opsi ini dimatikan secara default. Untuk mengizinkan lalu lintas IPv6 keluar, gunakan konsol atau `--vpc-config Ipv6AllowedForDualStack=true` opsi dengan perintah create-function atau update-function-configuration.

> **Note**
>
> Untuk memungkinkan lalu lintas IPv6 keluar di VPC, semua subnet yang terhubung ke fungsi harus subnet dual-stack. Lambda tidak mendukung koneksi IPv6 keluar untuk subnet IPv6-only di VPC atau koneksi IPv6 keluar untuk fungsi yang tidak terhubung ke VPC.

Anda dapat memperbarui kode fungsi Anda untuk secara eksplisit terhubung ke sumber daya subnet melalui IPv6. Contoh Python berikut membuka soket dan menghubungkan ke server IPv6.

Example— Connect ke server IPv6

```
def connect_to_server(event, context):
    server_address = event['host']
    server_port = event['port']
    message = event['message']
    run_connect_to_server(server_address, server_port, message)

def run_connect_to_server(server_address, server_port, message):
    sock = socket.socket(socket.AF_INET6, socket.SOCK_STREAM, 0)
    try:
        # Send data
        sock.connect((server_address, int(server_port), 0, 0))
        sock.sendall(message.encode())
        BUFF_SIZE = 4096
        data = b''
        while True:
            segment = sock.recv(BUFF_SIZE)
            data += segment
            # Either 0 or end of data
```

```
            if len(segment) < BUFF_SIZE:
                break
        return data
    finally:
        sock.close()
```

Praktik terbaik untuk menggunakan Lambda dengan Amazon VPC

Untuk memastikan bahwa konfigurasi VPC Lambda Anda memenuhi pedoman praktik terbaik, ikuti saran di bagian berikut.

Praktik terbaik keamanan

Untuk melampirkan fungsi Lambda Anda ke VPC, Anda harus memberikan peran eksekusi fungsi Anda sejumlah izin Amazon EC2. Izin ini diperlukan untuk membuat antarmuka jaringan yang digunakan fungsi Anda untuk mengakses sumber daya di VPC. Namun, izin ini juga secara implisit diberikan ke kode fungsi Anda. Ini berarti bahwa kode fungsi Anda memiliki izin untuk melakukan panggilan API Amazon EC2 ini.

Untuk mengikuti prinsip akses hak istimewa paling sedikit, tambahkan kebijakan penolakan seperti contoh berikut ke peran eksekusi fungsi Anda. Kebijakan ini mencegah kode fungsi Anda melakukan panggilan ke Amazon EC2 API, sambil tetap mengizinkan layanan Lambda mengelola sumber daya VPC atas nama Anda. Kebijakan menggunakan kunci `lambda:SourceFunctionArn` kondisi, yang hanya berlaku untuk panggilan API yang dibuat oleh kode fungsi Anda selama eksekusi. Untuk informasi selengkapnya, lihat the section called “Fungsi sumber ARN”.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "ec2:CreateNetworkInterface",
                "ec2:DeleteNetworkInterface",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DescribeSubnets",
                "ec2:DetachNetworkInterface",
                "ec2:AssignPrivateIpAddresses",
```

```
                "ec2:UnassignPrivateIpAddresses"
            ],
            "Resource": [ "*" ],
            "Condition": {
                "ArnEquals": {
                    "lambda:SourceFunctionArn": [
                        "arn:aws:lambda:us-
west-2:123456789012:function:my_function"
                    ]
                }
            }
        }
    ]
}
```

AWS menyediakan grup keamanan dan Daftar Kontrol Akses jaringan (ACL) untuk meningkatkan keamanan di VPC Anda. Grup keamanan mengontrol lalu lintas masuk dan keluar untuk sumber daya Anda, dan ACL jaringan mengontrol lalu lintas masuk dan keluar untuk subnet Anda. Grup keamanan menyediakan kontrol akses yang cukup untuk sebagian besar subnet. Anda dapat menggunakan ACL jaringan jika Anda menginginkan lapisan keamanan tambahan untuk VPC Anda. Untuk panduan umum tentang praktik terbaik keamanan saat menggunakan VPC Amazon, lihat Praktik terbaik keamanan untuk VPC Anda di Panduan Pengguna Amazon Virtual Private Cloud.

Praktik terbaik kinerja

Ketika Anda melampirkan fungsi Anda ke VPC, Lambda memeriksa untuk melihat apakah ada sumber daya jaringan yang tersedia (Hyperplane ENI) yang dapat digunakan untuk terhubung. Hyperplane ENI dikaitkan dengan kombinasi tertentu dari kelompok keamanan dan subnet VPC. Jika Anda telah melampirkan satu fungsi ke VPC, menentukan subnet dan grup keamanan yang sama saat Anda melampirkan fungsi lain berarti Lambda dapat berbagi sumber daya jaringan dan menghindari kebutuhan untuk membuat ENI Hyperplane baru. Untuk informasi selengkapnya tentang ENI Hyperplane dan siklus hidupnya, lihat. the section called “Memahami Antarmuka Jaringan Elastis Hyperplane (ENI)”

Memahami Antarmuka Jaringan Elastis Hyperplane (ENI)

Hyperplane ENI adalah sumber daya terkelola yang bertindak sebagai antarmuka jaringan antara fungsi Lambda Anda dan sumber daya yang Anda inginkan untuk terhubung ke fungsi Anda. Layanan Lambda membuat dan mengelola ENI ini secara otomatis saat Anda melampirkan fungsi Anda ke VPC.

Hyperplane ENIS tidak langsung terlihat oleh Anda, dan Anda tidak perlu mengonfigurasi atau mengelolanya. Namun, mengetahui cara kerjanya dapat membantu Anda memahami perilaku fungsi Anda saat Anda melampirkannya ke VPC.

Pertama kali Anda melampirkan fungsi ke VPC menggunakan subnet tertentu dan kombinasi grup keamanan, Lambda membuat Hyperplane ENI. Fungsi lain di akun Anda yang menggunakan kombinasi subnet dan grup keamanan yang sama juga dapat menggunakan ENI ini. Jika memungkinkan, Lambda menggunakan kembali ENI yang ada untuk mengoptimalkan pemanfaatan sumber daya dan meminimalkan pembuatan ENI baru. Setiap Hyperplane ENI mendukung hingga connections/ports 65.000. Jika jumlah koneksi melebihi batas ini, Lambda menskalakan jumlah ENI secara otomatis berdasarkan lalu lintas jaringan dan persyaratan konkurensi.

Untuk fungsi baru, saat Lambda membuat ENI Hyperplane, fungsi Anda tetap dalam status Pending dan Anda tidak dapat memanggilnya. Fungsi Anda beralih ke status Aktif hanya ketika Hyperplane ENI siap, yang dapat memakan waktu beberapa menit. Untuk fungsi yang ada, Anda tidak dapat melakukan operasi tambahan yang menargetkan fungsi, seperti membuat versi atau memperbarui kode fungsi, tetapi Anda dapat terus memanggil versi fungsi sebelumnya.

Sebagai bagian dari pengelolaan siklus hidup ENI, Lambda mungkin menghapus dan membuat ulang ENI untuk memuat lalu lintas jaringan keseimbangan di seluruh ENI atau untuk mengatasi masalah yang ditemukan dalam pemeriksaan kesehatan ENI. Selain itu, jika fungsi Lambda tetap menganggur selama 14 hari, Lambda merebut kembali ENI Hyperplane yang tidak digunakan dan menyetel status fungsi ke. `Inactive` Upaya pemanggilan berikutnya akan gagal, dan fungsi memasuki kembali status Pending hingga Lambda menyelesaikan pembuatan atau alokasi ENI Hyperplane. Kami menyarankan agar desain Anda tidak bergantung pada ketekunan ENI.

Saat Anda memperbarui fungsi untuk menghapus konfigurasi VPC-nya, Lambda membutuhkan waktu hingga 20 menit untuk menghapus Hyperplane ENI yang terpasang. Lambda hanya menghapus ENI jika tidak ada fungsi lain (atau versi fungsi yang diterbitkan) yang menggunakan Hyperplane ENI itu.

Lambda mengandalkan izin dalam peran eksekusi fungsi untuk menghapus ENI Hyperplane. Jika Anda menghapus peran eksekusi sebelum Lambda menghapus ENI Hyperplane, Lambda tidak akan dapat menghapus ENI Hyperplane. Anda dapat melakukan penghapusan secara manual.

Menggunakan kunci syarat IAM untuk pengaturan VPC

Anda dapat menggunakan tombol Lambda-specific kondisi untuk pengaturan VPC untuk memberikan kontrol izin tambahan untuk fungsi Lambda Anda. Misalnya, Anda dapat meminta semua fungsi

dalam organisasi Anda terhubung ke VPC. Anda juga dapat menentukan subnet dan grup keamanan yang dapat dan tidak dapat digunakan oleh pengguna fungsi.

Lambda mendukung kunci syarat berikut dalam kebijakan IAM:

- lambda: VpcIds — Izinkan atau tolak satu atau lebih VPC.
- lambda: SubnetIds — Izinkan atau tolak satu atau lebih subnet.
- lambda: SecurityGroupIds — Izinkan atau tolak satu atau lebih grup keamanan.

Operasi API Lambda CreateFunction dan UpdateFunctionConfiguration mendukung kunci kondisi ini. Untuk informasi selengkapnya tentang penggunaan kunci syarat dalam kebijakan IAM, lihat elemen kebijakan IAM JSON: Syarat di Panduan Pengguna IAM.

> **Tip**
>
> Jika fungsi Anda sudah mencakup konfigurasi VPC dari permintaan API sebelumnya, Anda dapat mengirim permintaan `UpdateFunctionConfiguration` tanpa konfigurasi VPC.

Contoh kebijakan dengan kunci syarat untuk pengaturan VPC

Contoh-contoh berikut ini menunjukkan cara menggunakan kunci syarat untuk pengaturan VPC. Setelah Anda membuat pernyataan kebijakan dengan batasan yang diinginkan, tambahkan pernyataan kebijakan untuk pengguna atau peran target.

Pastikan bahwa pengguna hanya menerapkan fungsi VPC-connected

Untuk memastikan bahwa semua pengguna hanya menerapkan VPC-connected fungsi, Anda dapat menolak operasi pembuatan dan pembaruan fungsi yang tidak menyertakan ID VPC yang valid.

Perhatikan bahwa ID VPC bukan parameter input ke `CreateFunction` atau permintaan `UpdateFunctionConfiguration`. Lambda mengambil nilai ID VPC berdasarkan parameter subnet dan grup keamanan.

JSON

```
{
  "Version":"2012-10-17",
```

```
  "Statement": [
    {
      "Sid": "EnforceVPCFunction",
      "Action": [
          "lambda:CreateFunction",
          "lambda:UpdateFunctionConfiguration"
       ],
      "Effect": "Deny",
      "Resource": "*",
      "Condition": {
        "Null": {
           "lambda:VpcIds": "true"
        }
      }
    }
  ]
}
```

Menolak akses pengguna ke VPC, subnet, atau grup keamanan tertentu

Untuk menolak akses pengguna ke VPC tertentu, gunakan `StringEquals` untuk memeriksa nilai syarat `lambda:VpcIds`. Contoh berikut menolak akses pengguna ke `vpc-1` dan `vpc-2`.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "EnforceOutOfVPC",
            "Action": [
                "lambda:CreateFunction",
                "lambda:UpdateFunctionConfiguration"
            ],
            "Effect": "Deny",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "lambda:VpcIds": [
                        "vpc-1",
                        "vpc-2"
```

```
                    ]
                }
            }
        }
    ]
}
```

Untuk menolak akses pengguna ke subnet tertentu, gunakan `StringEquals` untuk memeriksa nilai syarat `lambda:SubnetIds`. Contoh berikut menolak akses pengguna ke `subnet-1` dan `subnet-2`.

```
{
      "Sid": "EnforceOutOfSubnet",
      "Action": [
          "lambda:CreateFunction",
          "lambda:UpdateFunctionConfiguration"
       ],
      "Effect": "Deny",
      "Resource": "*",
      "Condition": {
        "ForAnyValue:StringEquals": {
            "lambda:SubnetIds": ["subnet-1", "subnet-2"]
        }
      }
    }
```

Untuk menolak akses pengguna ke grup keamanan tertentu, gunakan `StringEquals` untuk memeriksa nilai syarat `lambda:SecurityGroupIds`. Contoh berikut menolak akses pengguna ke `sg-1` dan `sg-2`.

```
{
      "Sid": "EnforceOutOfSecurityGroups",
      "Action": [
          "lambda:CreateFunction",
          "lambda:UpdateFunctionConfiguration"
       ],
      "Effect": "Deny",
      "Resource": "*",
      "Condition": {
        "ForAnyValue:StringEquals": {
```

```
            "lambda:SecurityGroupIds": ["sg-1", "sg-2"]
        }
      }
    }
  ]
}
```

Mengizinkan pengguna untuk membuat dan memperbarui fungsi dengan pengaturan VPC tertentu

Untuk mengizinkan pengguna mengakses VPC tertentu, gunakan `StringEquals` untuk memeriksa nilai syarat `lambda:VpcIds`. Contoh berikut mengizinkan pengguna mengakses `vpc-1` dan `vpc-2`.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "EnforceStayInSpecificVpc",
            "Action": [
                "lambda:CreateFunction",
                "lambda:UpdateFunctionConfiguration"
            ],
            "Effect": "Allow",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "lambda:VpcIds": [
                        "vpc-1",
                        "vpc-2"
                    ]
                }
            }
        }
    ]
}
```

Untuk mengizinkan pengguna mengakses subnet tertentu, gunakan `StringEquals` untuk memeriksa nilai syarat `lambda:SubnetIds`. Contoh berikut mengizinkan pengguna mengakses `subnet-1` dan `subnet-2`.

```
{
      "Sid": "EnforceStayInSpecificSubnets",
      "Action": [
          "lambda:CreateFunction",
          "lambda:UpdateFunctionConfiguration"
       ],
      "Effect": "Allow",
      "Resource": "*",
      "Condition": {
        "ForAllValues:StringEquals": {
            "lambda:SubnetIds": ["subnet-1", "subnet-2"]
        }
      }
    }
```

Untuk mengizinkan pengguna mengakses grup keamanan tertentu, gunakan `StringEquals` untuk memeriksa nilai syarat `lambda:SecurityGroupIds`. Contoh berikut mengizinkan pengguna mengakses `sg-1` dan `sg-2`.

```
{
      "Sid": "EnforceStayInSpecificSecurityGroup",
      "Action": [
          "lambda:CreateFunction",
          "lambda:UpdateFunctionConfiguration"
       ],
      "Effect": "Allow",
      "Resource": "*",
      "Condition": {
        "ForAllValues:StringEquals": {
            "lambda:SecurityGroupIds": ["sg-1", "sg-2"]
        }
      }
    }
  ]
}
```

Tutorial VPC

Dalam tutorial berikut, Anda menghubungkan fungsi Lambda ke sumber daya di VPC Anda.

- Tutorial: Menggunakan fungsi Lambda untuk mengakses Amazon RDS di Amazon VPC
- Tutorial: Mengonfigurasi fungsi Lambda untuk mengakses Amazon di ElastiCache VPC Amazon

Untuk menghubungkan fungsi Anda ke sumber daya di VPC milik orang lain Akun AWS, lihat. the section called “Melampirkan fungsi ke sumber daya di akun lain”

Memberikan Lambda Event Functions akses ke sumber daya di Amazon VPC di akun lain

Anda dapat memberikan AWS Lambda fungsi Anda akses ke sumber daya di Amazon VPC di Amazon Virtual Private Cloud yang dikelola oleh akun lain, tanpa mengekspos VPC ke internet. Dengan pola akses ini, Anda dapat berbagi data dengan organisasi lain menggunakan AWS. Dengan menggunakan pola akses ini, Anda dapat berbagi data antara VPC dengan tingkat keamanan dan kinerja yang lebih besar daripada melalui internet. Konfigurasikan fungsi Lambda Anda untuk menggunakan koneksi peering VPC Amazon untuk mengakses sumber daya ini.

> **Warning**
>
> Saat Anda mengizinkan akses antar akun atau VPC, periksa apakah paket Anda memenuhi persyaratan keamanan masing-masing organisasi yang mengelola akun ini. Mengikuti instruksi dalam dokumen ini memengaruhi postur keamanan sumber daya Anda.

Dalam tutorial ini, Anda menghubungkan dua akun bersama dengan koneksi peering menggunakan IPv4. Anda mengonfigurasi fungsi Lambda yang belum terhubung ke VPC Amazon. Anda mengonfigurasi resolusi DNS untuk menghubungkan fungsi Anda ke sumber daya yang tidak menyediakan IP statis. Untuk menyesuaikan instruksi ini dengan skenario peering lainnya, lihat VPC Peering Guide.

Prasyarat

Untuk memberikan akses fungsi Lambda ke sumber daya di acccount lain, Anda harus memiliki:

- Fungsi Lambda, dikonfigurasi untuk mengautentikasi dengan dan kemudian membaca dari sumber daya Anda.
- Sumber daya di akun lain, seperti cluster Amazon RDS, tersedia melalui Amazon VPC.
- Kredensil untuk akun fungsi Lambda Anda dan akun sumber daya Anda. Jika Anda tidak berwenang untuk menggunakan akun sumber daya Anda, hubungi pengguna yang berwenang untuk menyiapkan akun itu.
- Izin untuk membuat dan memperbarui VPC (dan mendukung sumber daya Amazon VPC) untuk dikaitkan dengan fungsi Lambda Anda.
- Izin untuk memperbarui peran eksekusi dan konfigurasi VPC untuk fungsi Lambda Anda.

- Izin untuk membuat koneksi peering VPC di akun fungsi Lambda Anda.
- Izin untuk menerima koneksi peering VPC di akun sumber daya Anda.
- Izin untuk memperbarui konfigurasi VPC sumber daya Anda (dan mendukung sumber daya VPC Amazon).
- Izin untuk menjalankan fungsi Lambda Anda.

Buat VPC Amazon di akun fungsi Anda

Buat VPC Amazon, subnet, tabel rute, dan grup keamanan di akun fungsi Lambda Anda.

Untuk membuat VPC, subnet, dan sumber daya VPC lainnya menggunakan konsol

1. Buka Konsol VPC Amazon di https://console.aws.amazon.com/vpc/.
2. Di dasbor, pilih Buat VPC.
3. Untuk blok IPv4 CIDR, sediakan blok CIDR pribadi. Blok CIDR Anda tidak boleh tumpang tindih dengan blok yang digunakan dalam VPC sumber daya Anda. Jangan memilih blok yang digunakan VPC sumber daya Anda untuk menetapkan IP ke sumber daya atau blok yang sudah ditentukan dalam tabel rute di VPC sumber daya Anda. Untuk informasi lebih lanjut tentang mendefinisikan blok CIDR yang sesuai, lihat blok VPC CIDR.
4. Pilih Sesuaikan AZ.
5. Pilih AZ yang sama dengan sumber daya Anda.
6. Untuk Jumlah subnet publik, pilih 0.
7. Untuk titik akhir VPC, pilih Tidak Ada.
8. Pilih Buat VPC.

Berikan izin VPC ke peran eksekusi fungsi Anda

Lampirkan AWSLambdaVPCAccessExecutionRoleke peran eksekusi fungsi Anda untuk memungkinkannya terhubung ke VPC.

Untuk memberikan izin VPC ke peran eksekusi fungsi Anda

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih nama fungsi Anda.
3. Pilih Konfigurasi.

4. Pilih Izin.
5. Di bawah Nama peran, pilih peran eksekusi.
6. Di bagian Kebijakan izin, pilih Tambahkan izin.
7. Dalam daftar dropdown, pilih Lampirkan kebijakan.
8. Dalam kotak pencarian, masukkan `AWSLambdaVPCAccessExecutionRole`.
9. Di sebelah kiri nama kebijakan, pilih kotak centang.
10. Pilih Tambahkan izin.

Untuk melampirkan fungsi Anda ke VPC Amazon Anda

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih nama fungsi Anda.
3. Pilih tab Konfigurasi, lalu pilih VPC.
4. Pilih Edit.
5. Di bawah VPC, pilih VPC Anda
6. Di bawah Subnet, pilih subnet Anda.
7. Di bawah Grup keamanan, pilih grup keamanan default untuk VPC Anda.
8. Pilih Simpan.

Buat permintaan koneksi peering VPC

Buat permintaan koneksi peering VPC dari VPC fungsi Anda (VPC pemohon) ke VPC sumber daya Anda (VPC penerima).

Untuk meminta koneksi peering VPC dari VPC fungsi Anda

1. Buka https://console.aws.amazon.com/vpc/.
2. Di panel navigasi, pilih Koneksi peering.
3. Pilih Buat koneksi peering.
4. Untuk ID VPC (Pemohon), pilih VPC fungsi Anda.
5. Untuk ID Akun, masukkan ID akun sumber daya Anda.
6. Untuk ID VPC (Penerima), masukkan VPC sumber daya Anda.

Siapkan akun sumber daya Anda

Untuk membuat koneksi peering Anda dan menyiapkan VPC sumber daya Anda untuk menggunakan koneksi, masuk ke akun sumber daya Anda dengan peran yang memegang izin yang tercantum dalam prasyarat. Langkah-langkah untuk masuk mungkin berbeda berdasarkan bagaimana akun diamankan. Untuk informasi selengkapnya tentang cara masuk ke AWS akun, lihat Panduan AWS Sign-in Pengguna. Di akun sumber daya Anda, lakukan prosedur berikut.

Untuk menerima permintaan koneksi peering VPC

1. Buka https://console.aws.amazon.com/vpc/.
2. Di panel navigasi, pilih Koneksi peering.
3. Pilih koneksi peering VPC yang tertunda (statusnya adalah penerimaan tertunda).
4. Pilih Tindakan
5. Dari daftar dropdown, pilih Terima permintaan.
6. Saat diminta konfirmasi, pilih Terima permintaan.
7. Pilih Ubah tabel rute saya sekarang untuk menambahkan rute ke tabel rute utama untuk VPC Anda sehingga Anda dapat mengirim dan menerima lalu lintas melintasi koneksi peering.

Periksa tabel rute untuk VPC sumber daya. Rute yang dihasilkan oleh Amazon VPC mungkin tidak membangun konektivitas, berdasarkan cara VPC sumber daya Anda diatur. Periksa konflik antara rute baru dan konfigurasi yang ada untuk VPC. Untuk informasi selengkapnya tentang pemecahan masalah, lihat Memecahkan masalah koneksi peering VPC di Panduan Peering VPC Amazon Virtual Private Cloud.

Untuk memperbarui grup keamanan untuk sumber daya Anda

1. Buka https://console.aws.amazon.com/vpc/.
2. Pada panel navigasi, pilih Grup keamanan.
3. Pilih grup keamanan untuk sumber daya Anda.
4. Pilih Tindakan.
5. Dari daftar dropdown, pilih Edit aturan masuk.
6. Pilih Tambahkan aturan.
7. Untuk Sumber, masukkan ID akun dan ID grup keamanan fungsi Anda, dipisahkan dengan garis miring ke depan (misalnya, 111122223333/sg -1a2b3c4d).

8. Pilih Edit aturan keluar.
9. Periksa apakah lalu lintas keluar dibatasi. Pengaturan VPC default memungkinkan semua lalu lintas keluar. Jika lalu lintas keluar dibatasi, lanjutkan ke langkah berikutnya.
10. Pilih Tambahkan aturan.
11. Untuk Tujuan masukkan ID akun dan ID grup keamanan fungsi Anda, dipisahkan dengan garis miring ke depan (misalnya, 111122223333/sg -1a2b3c4d).
12. Pilih Simpan aturan.

Untuk mengaktifkan resolusi DNS untuk koneksi peering Anda

1. Buka https://console.aws.amazon.com/vpc/.
2. Di panel navigasi, pilih Koneksi peering.
3. Pilih koneksi peering Anda.
4. Pilih Tindakan.
5. Pilih Edit pengaturan DNS.
6. Di bawah resolusi DNS Accepter, pilih Izinkan VPC pemohon untuk menyelesaikan DNS host VPC penerima ke IP pribadi.
7. Pilih Simpan perubahan.

Perbarui konfigurasi VPC di akun fungsi Anda

Masuk ke akun fungsi Anda, lalu perbarui konfigurasi VPC.

Untuk menambahkan rute untuk koneksi peering VPC Anda

1. Buka https://console.aws.amazon.com/vpc/.
2. Di panel navigasi, pilih Tabel rute.
3. Pilih kotak centang di sebelah nama tabel rute untuk subnet yang terkait dengan fungsi Anda.
4. Pilih Tindakan.
5. Pilih Edit rute.
6. Pilih Tambahkan rute.
7. Untuk Tujuan, masukkan blok CIDR untuk VPC sumber daya Anda.
8. Untuk Target, pilih koneksi peering VPC Anda.

9. Pilih Simpan perubahan.

Untuk informasi selengkapnya tentang pertimbangan yang mungkin Anda temui saat memperbarui tabel rute, lihat Perbarui tabel rute Anda untuk koneksi peering VPC.

Untuk memperbarui grup keamanan untuk fungsi Lambda Anda

1. Buka https://console.aws.amazon.com/vpc/.
2. Pada panel navigasi, pilih Grup keamanan.
3. Pilih Tindakan.
4. Pilih Edit aturan masuk.
5. Pilih Tambahkan aturan.
6. Untuk Sumber, masukkan ID akun sumber daya dan ID grup keamanan Anda, dipisahkan dengan garis miring ke depan (misalnya, 111122223333/sg -1a2b3c4d).
7. Pilih Simpan aturan.

Untuk mengaktifkan resolusi DNS untuk koneksi peering Anda

1. Buka https://console.aws.amazon.com/vpc/.
2. Di panel navigasi, pilih Koneksi peering.
3. Pilih koneksi peering Anda.
4. Pilih Tindakan.
5. Pilih Edit pengaturan DNS.
6. Di bawah resolusi DNS Pemohon, pilih Izinkan VPC penerima untuk menyelesaikan DNS host VPC pemohon ke IP pribadi.
7. Pilih Simpan perubahan.

Uji fungsi Anda

Untuk membuat acara pengujian dan memeriksa output fungsi Anda

1. Di panel Sumber kode, pilih Uji.
2. Pilih Buat acara baru.
3. Di panel Event JSON, ganti nilai default dengan input yang sesuai untuk fungsi Lambda Anda.

4. Pilih Panggil.
5. Di tab Hasil eksekusi, konfirmasikan bahwa Respons berisi output yang Anda harapkan.

Selain itu, Anda dapat memeriksa log fungsi Anda untuk memverifikasi log seperti yang Anda harapkan.

Untuk melihat catatan pemanggilan fungsi Anda di Log CloudWatch

1. Pilih tab Pantau.
2. Pilih Lihat CloudWatch log.
3. Di tab Log streams, pilih aliran log untuk pemanggilan fungsi Anda.
4. Konfirmasikan log Anda seperti yang Anda harapkan.

Aktifkan akses internet untuk Fungsi VPC-connected Acara Lambda

Secara default, fungsi Lambda berjalan di Lambda-managed VPC yang memiliki akses internet. Untuk mengakses sumber daya dalam VPC di akun Anda, Anda dapat menambahkan konfigurasi VPC ke suatu fungsi. Ini membatasi fungsi untuk sumber daya dalam VPC itu, kecuali VPC memiliki akses internet. Halaman ini menjelaskan cara menyediakan akses internet ke fungsi VPC-connected Lambda.

Saya belum memiliki VPC

Buat VPC

Alur kerja Buat VPC menciptakan semua sumber daya VPC yang diperlukan untuk fungsi Lambda untuk mengakses internet publik dari subnet pribadi, termasuk subnet, gateway NAT, gateway internet, dan entri tabel rute.

Untuk membuat VPC

1. Buka konsol Amazon VPC di. https://console.aws.amazon.com/vpc/
2. Di dasbor, pilih Buat VPC.
3. Agar Sumber Daya dapat dibuat, pilih VPC dan lainnya.
4. Konfigurasikan VPC
    a. Untuk Pembuatan otomatis tanda nama, masukkan nama untuk VPC.
    b. Untuk blok IPv4 CIDR, Anda dapat menyimpan saran default, atau sebagai alternatif Anda dapat memasukkan blok CIDR yang diperlukan oleh aplikasi atau jaringan Anda.
    c. Jika aplikasi Anda berkomunikasi dengan menggunakan alamat IPv6, pilih blok IPv6 CIDR, blok IPv6 CIDR. Amazon-provided
5. Konfigurasikan subnet
    a. Untuk Jumlah Availability Zone, pilih 2. Kami merekomendasikan setidaknya dua AZ untuk ketersediaan tinggi.
    b. Untuk Jumlah subnet publik, pilih 2.
    c. Untuk Jumlah subnet pribadi, pilih 2.
    d. Anda dapat menyimpan blok CIDR default untuk subnet publik, atau sebagai alternatif Anda dapat memperluas Kustomisasi blok CIDR subnet dan memasukkan blok CIDR. Untuk informasi selengkapnya, lihat Blok CIDR Subnet.

6. Untuk gateway NAT, pilih 1 per AZ untuk meningkatkan ketahanan.
7. Untuk gateway internet Egress saja, pilih Ya jika Anda memilih untuk menyertakan blok CIDR IPv6.
8. Untuk titik akhir VPC, pertahankan default (S3 Gateway). Tidak ada biaya untuk opsi ini. Untuk informasi selengkapnya, lihat Jenis titik akhir VPC untuk Amazon S3.
9. Untuk opsi DNS, pertahankan pengaturan default.
10. Pilih Buat VPC.

Konfigurasikan fungsi Lambda

Untuk mengonfigurasi VPC saat Anda membuat fungsi

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih Buat Fungsi Acara.
3. Di bawah Informasi Dasar, untuk Nama fungsi, masukkan nama untuk fungsi Anda.
4. Perluas Pengaturan lanjutan.
5. Pilih Aktifkan VPC, lalu pilih VPC.
6. (Opsional) Untuk mengizinkan lalu lintas IPv6 keluar, pilih Izinkan lalu lintas IPv6 untuk subnet dual-stack.
7. Untuk Subnet, pilih semua subnet pribadi. Subnet pribadi dapat mengakses internet melalui gateway NAT. Menghubungkan fungsi ke subnet publik tidak memberikan akses internet.

   > **Note**
   >
   > Jika Anda memilih Izinkan lalu lintas IPv6 untuk subnet dual-stack, semua subnet yang dipilih harus memiliki blok CIDR IPv4 dan blok CIDR IPv6.

8. Untuk grup Keamanan, pilih grup keamanan yang memungkinkan lalu lintas keluar.
9. Pilih Buat Fungsi Acara.

Lambda secara otomatis membuat peran eksekusi dengan kebijakan AWSLambdaVPCAccessExecutionRole AWS terkelola. Izin dalam kebijakan ini hanya diperlukan untuk membuat antarmuka jaringan elastis untuk konfigurasi VPC, bukan untuk menjalankan fungsi Anda. Untuk menerapkan izin hak istimewa paling sedikit, Anda dapat menghapus

AWSLambdaVPCAccessExecutionRolekebijakan dari peran eksekusi setelah membuat fungsi dan konfigurasi VPC. Untuk informasi selengkapnya, lihat Izin IAM yang diperlukan.

Untuk mengonfigurasi VPC untuk fungsi yang ada

Untuk menambahkan konfigurasi VPC ke fungsi yang ada, peran eksekusi fungsi harus memiliki izin untuk membuat dan mengelola antarmuka jaringan elastis. Kebijakan AWSLambdaVPCAccessExecutionRole AWS terkelola mencakup izin yang diperlukan. Untuk menerapkan izin hak istimewa paling sedikit, Anda dapat menghapus AWSLambdaVPCAccessExecutionRolekebijakan dari peran eksekusi setelah membuat konfigurasi VPC.

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih fungsi.
3. Pilih tab Konfigurasi, lalu pilih VPC.
4. Di bagian VPC, pilih Edit.
5. Pilih VPC.
6. (Opsional) Untuk mengizinkan lalu lintas IPv6 keluar, pilih Izinkan lalu lintas IPv6 untuk subnet dual-stack.
7. Untuk Subnet, pilih semua subnet pribadi. Subnet pribadi dapat mengakses internet melalui gateway NAT. Menghubungkan fungsi ke subnet publik tidak memberikan akses internet.

   > **Note**
   >
   > Jika Anda memilih Izinkan lalu lintas IPv6 untuk subnet dual-stack, semua subnet yang dipilih harus memiliki blok CIDR IPv4 dan blok CIDR IPv6.

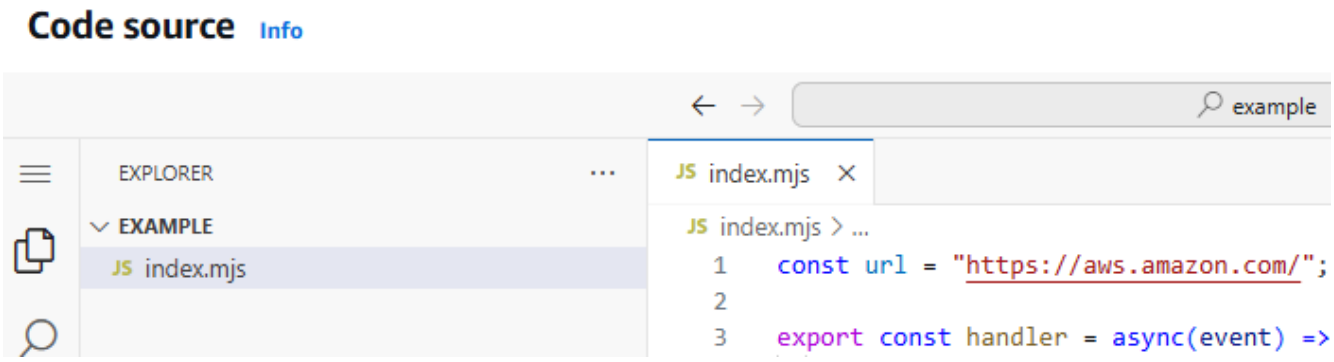
8. Untuk grup Keamanan, pilih grup keamanan yang memungkinkan lalu lintas keluar.
9. Pilih Simpan.

Uji fungsi

Gunakan kode contoh berikut untuk mengonfirmasi bahwa VPC-connected fungsi Anda dapat menjangkau internet publik. Jika berhasil, kode mengembalikan kode `200` status. Jika tidak berhasil, fungsi akan habis.

Node.js

1. Di panel Sumber kode di konsol Lambda, tempel kode berikut ke dalam file index.mjs. Fungsi ini membuat permintaan HTTP GET ke titik akhir publik dan mengembalikan kode respons HTTP untuk menguji apakah fungsi tersebut memiliki akses ke internet publik.

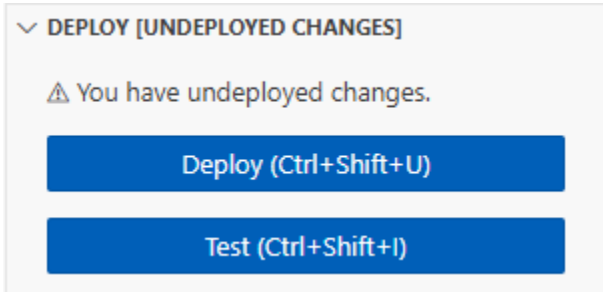


Example— Permintaan HTTP dengan async/await

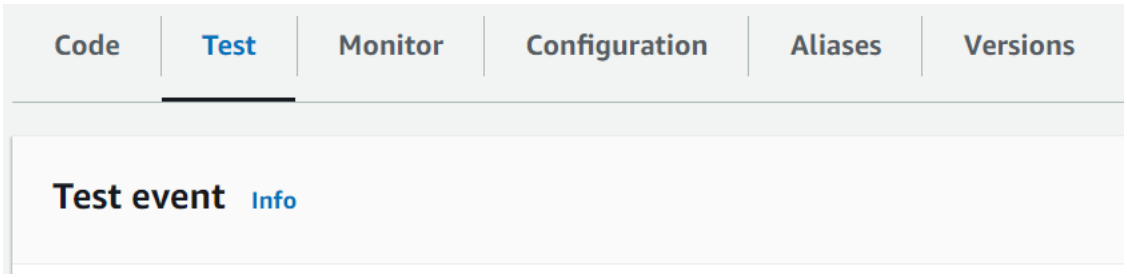
```
const url = "https://aws.amazon.com/";

export const handler = async(event) => {
    try {
        const res = await fetch(url);
        console.info("status", res.status);
        return res.status;
    }
    catch (e) {
        console.error(e);
        return 500;
    }
};
```

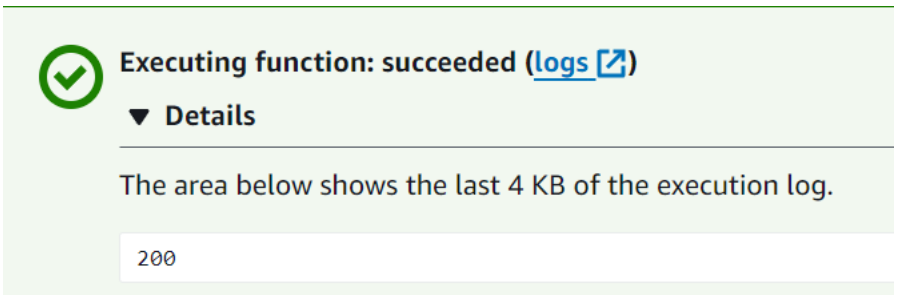
2. Di bagian DEPLOY, pilih Deploy untuk memperbarui kode fungsi Anda:



3. Pilih tab Uji.



4. Pilih Uji.
5. Fungsi mengembalikan kode `200` status. Ini berarti bahwa fungsi tersebut memiliki akses internet keluar.

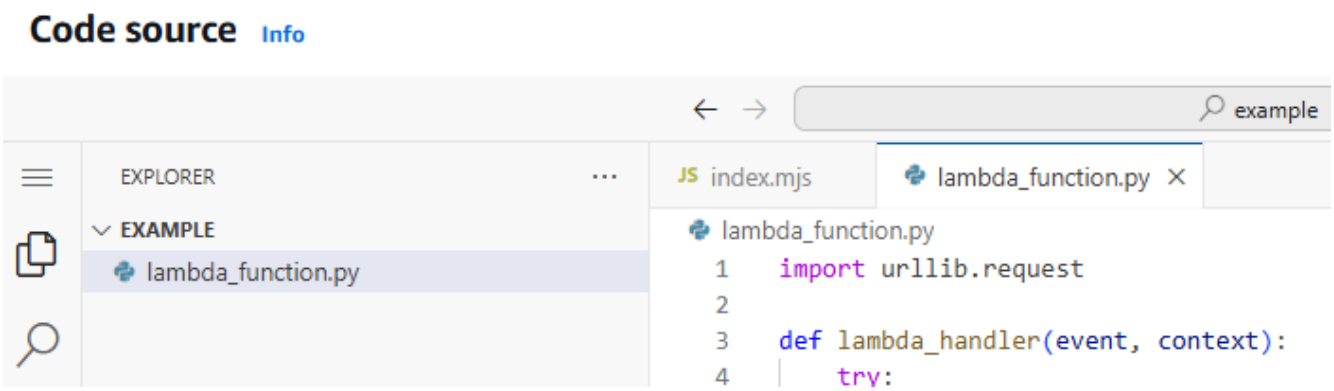


Jika fungsi tidak dapat menjangkau internet publik, Anda mendapatkan pesan kesalahan seperti ini:

```
{
  "errorMessage": "2024-04-11T17:22:20.857Z abe12jlc-640a-8157-0249-9be825c2y110
 Task timed out after 3.01 seconds"
}
```

Python

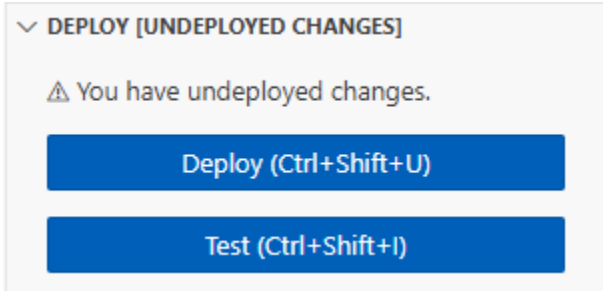
1. Di panel Sumber kode di konsol Lambda, tempel kode berikut ke dalam file lambda_function.py. Fungsi ini membuat permintaan HTTP GET ke titik akhir publik dan mengembalikan kode respons HTTP untuk menguji apakah fungsi tersebut memiliki akses ke internet publik.



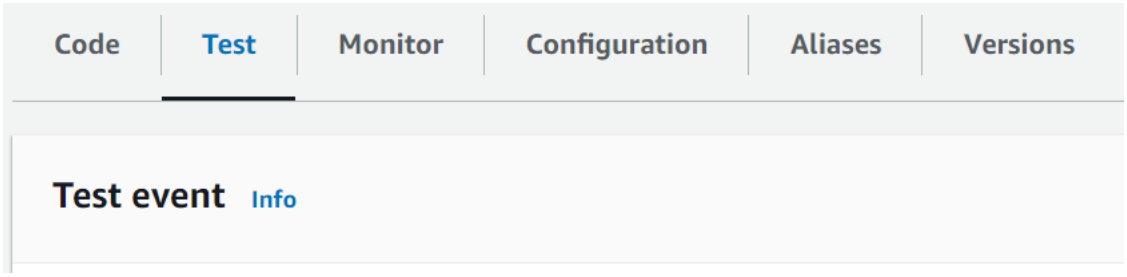

```
import urllib.request

def lambda_handler(event, context):
    try:
        response = urllib.request.urlopen('https://aws.amazon.com')
        status_code = response.getcode()
        print('Response Code:', status_code)
        return status_code
    except Exception as e:
        print('Error:', e)
        raise e
```

2. Di bagian DEPLOY, pilih Deploy untuk memperbarui kode fungsi Anda:

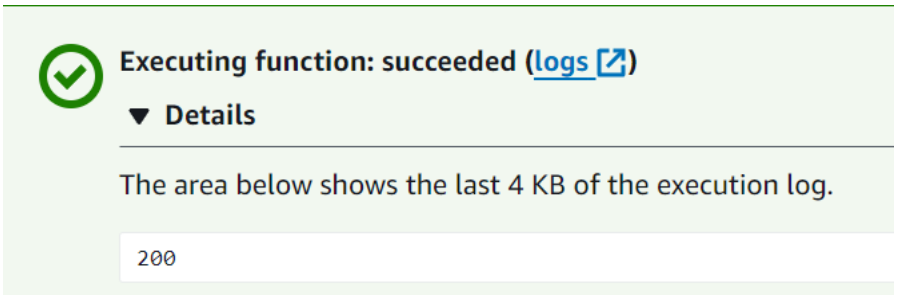


3. Pilih tab Uji.



4. Pilih Uji.

5. Fungsi mengembalikan kode `200` status. Ini berarti bahwa fungsi tersebut memiliki akses internet keluar.



Jika fungsi tidak dapat menjangkau internet publik, Anda mendapatkan pesan kesalahan seperti ini:

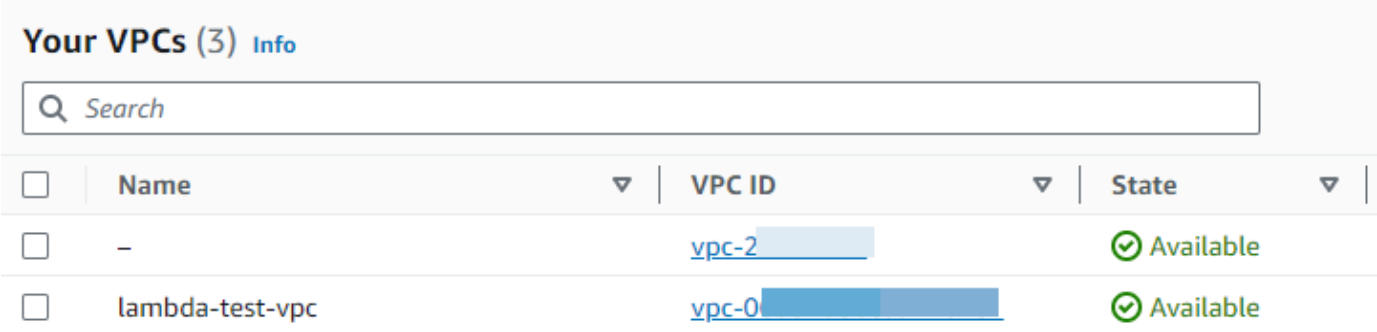
```
{
  "errorMessage": "2024-04-11T17:22:20.857Z abe12jlc-640a-8157-0249-9be825c2y110
 Task timed out after 3.01 seconds"
}
```

Saya sudah memiliki VPC

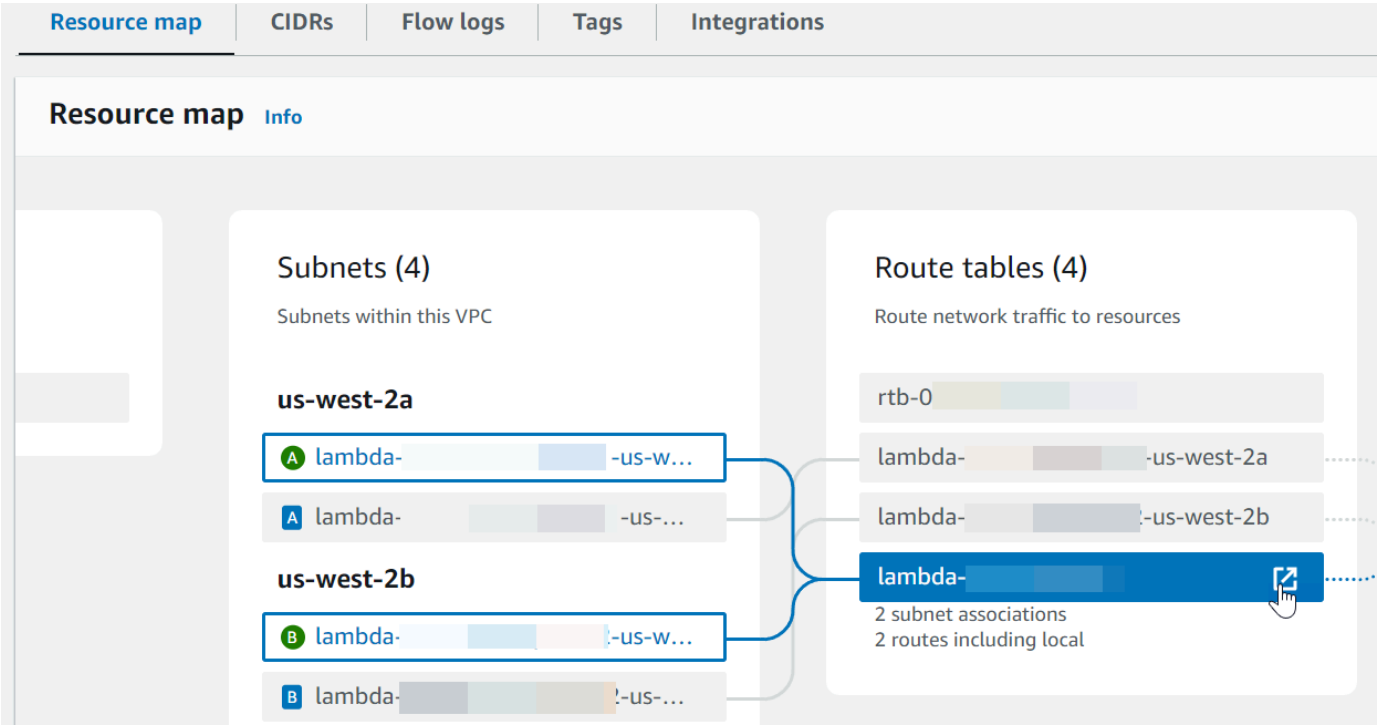
Jika Anda sudah memiliki VPC tetapi Anda perlu mengonfigurasi akses internet publik untuk fungsi Lambda, ikuti langkah-langkah ini. Prosedur ini mengasumsikan bahwa VPC Anda memiliki setidaknya dua subnet. Jika Anda tidak memiliki dua subnet, lihat Membuat subnet di Panduan Pengguna Amazon VPC.

Verifikasi konfigurasi tabel rute

1. Buka konsol Amazon VPC di. https://console.aws.amazon.com/vpc/
2. Pilih ID VPC.



3. Gulir ke bawah ke bagian Peta sumber daya. Perhatikan pemetaan tabel rute. Buka setiap tabel rute yang dipetakan ke subnet.

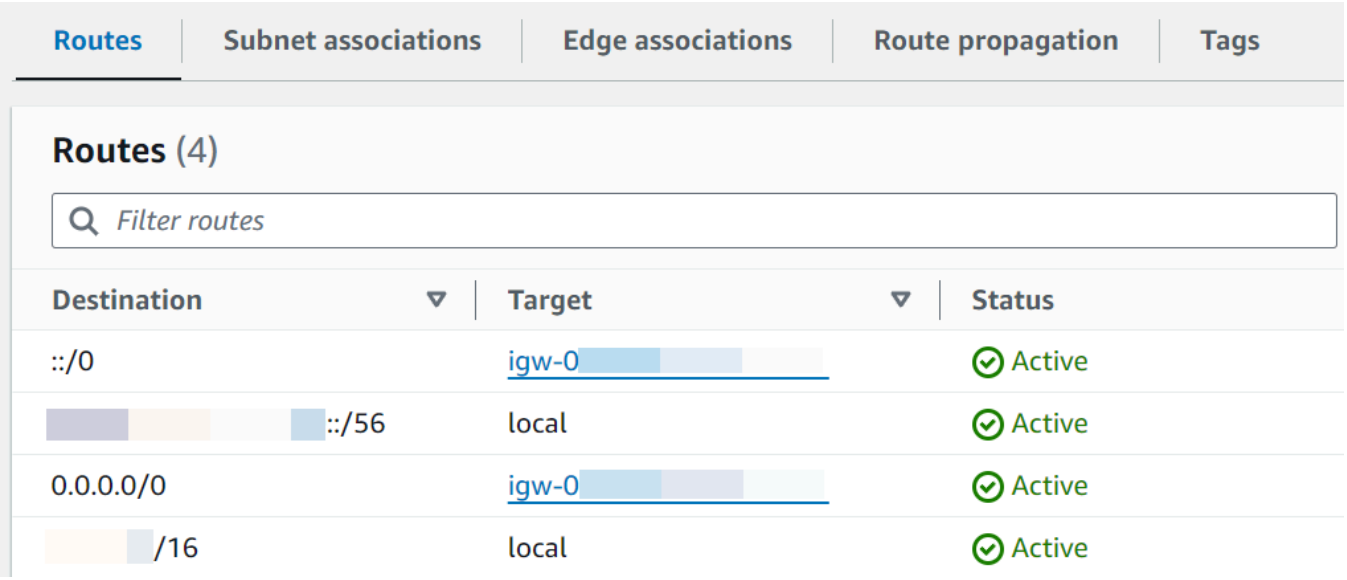


4. Gulir ke bawah ke tab Rute. Tinjau rute untuk menentukan apakah VPC Anda memiliki kedua tabel rute berikut. Masing-masing persyaratan ini harus dipenuhi oleh tabel rute terpisah.

   - Internet-bound lalu lintas (`0.0.0.0/0`untuk IPv4, `::/0` untuk IPv6) dirutekan ke gateway internet (). `igw-xxxxxxxxxx` Ini berarti bahwa subnet yang terkait dengan tabel rute adalah subnet publik.

     > **Note**
     >
     > Jika subnet Anda tidak memiliki blok IPv6 CIDR, Anda hanya akan melihat rute IPv4 (). `0.0.0.0/0`

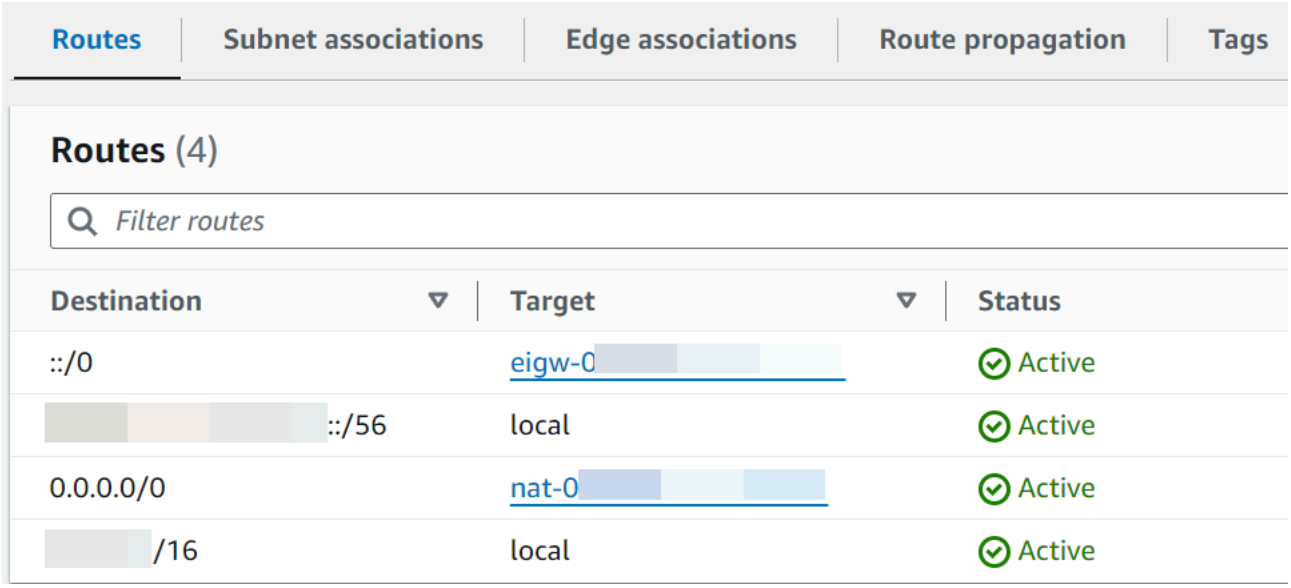
Example tabel rute subnet publik



- Internet-bound lalu lintas untuk IPv4 (`0.0.0.0/0`) dirutekan ke gateway NAT (`nat-xxxxxxxxxx`) yang terkait dengan subnet publik. Ini berarti bahwa subnet adalah subnet pribadi yang dapat mengakses internet melalui gateway NAT.

> **Note**
>
> Jika subnet Anda memiliki blok IPv6 CIDR, tabel rute juga harus merutekan lalu lintas IPv6 yang terikat internet () ke gateway internet khusus keluar (`::/0`). `eigw-xxxxxxxxxx` Jika subnet Anda tidak memiliki blok IPv6 CIDR, Anda hanya akan melihat rute IPv4 (). `0.0.0.0/0`

Example tabel rute subnet pribadi



5. Ulangi langkah sebelumnya hingga Anda meninjau setiap tabel rute yang terkait dengan subnet di VPC Anda dan mengonfirmasi bahwa Anda memiliki tabel rute dengan gateway internet dan tabel rute dengan gateway NAT.

   Jika Anda tidak memiliki dua tabel rute, satu dengan rute ke gateway internet dan satu dengan rute ke gateway NAT, ikuti langkah-langkah ini untuk membuat sumber daya dan entri tabel rute yang hilang.

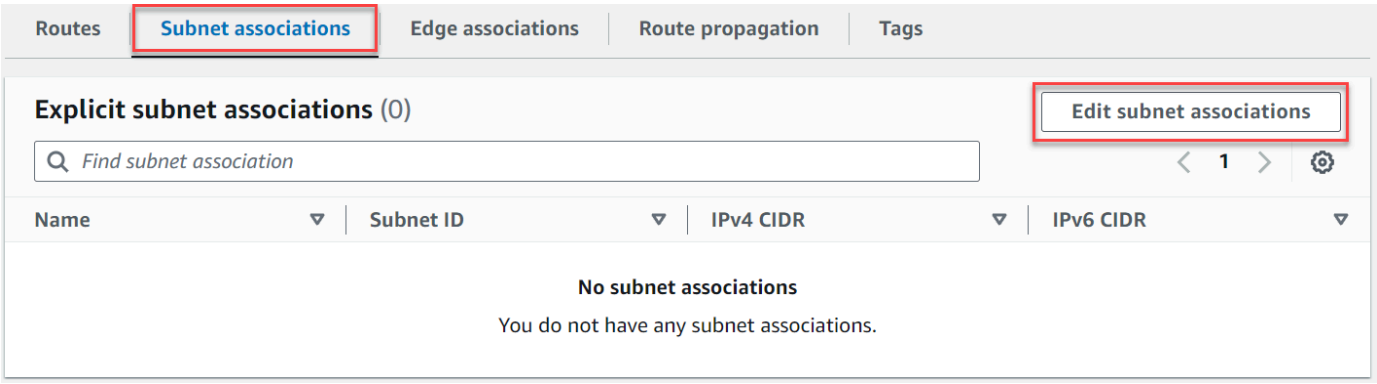
Membuat tabel rute

Ikuti langkah-langkah ini untuk membuat tabel rute dan mengaitkannya dengan subnet.

Untuk membuat tabel rute khusus menggunakan konsol Amazon VPC

1. Buka konsol Amazon VPC di. https://console.aws.amazon.com/vpc/
2. Di panel navigasi, pilih Tabel rute.
3. Pilih Buat tabel rute.
4. (Opsional) Untuk Nama, masukkan nama untuk tabel rute Anda.
5. Untuk VPC, pilih VPC Anda.
6. (Opsional) Untuk menambahkan tag, pilih Tambahkan tag baru dan masukkan kunci tag dan nilai tag.

7. Pilih Buat tabel rute.
8. Pada tab Pengaitan subnet, pilih Sunting pengaitan subnet.



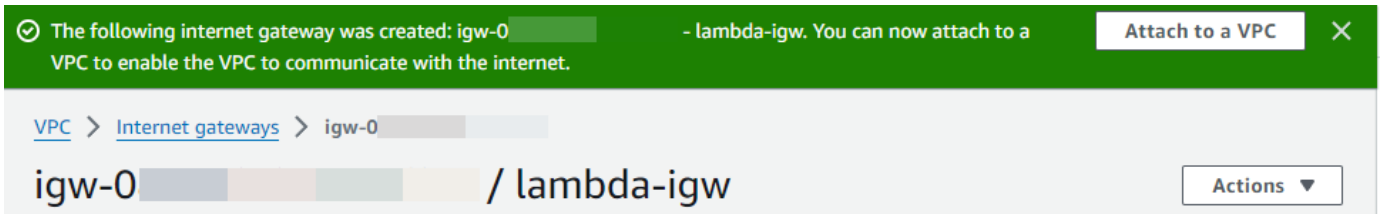
9. Pilih kotak centang untuk subnet untuk dikaitkan dengan tabel rute.
10. Pilih Simpan pengaitan.

Membuat gateway internet baru

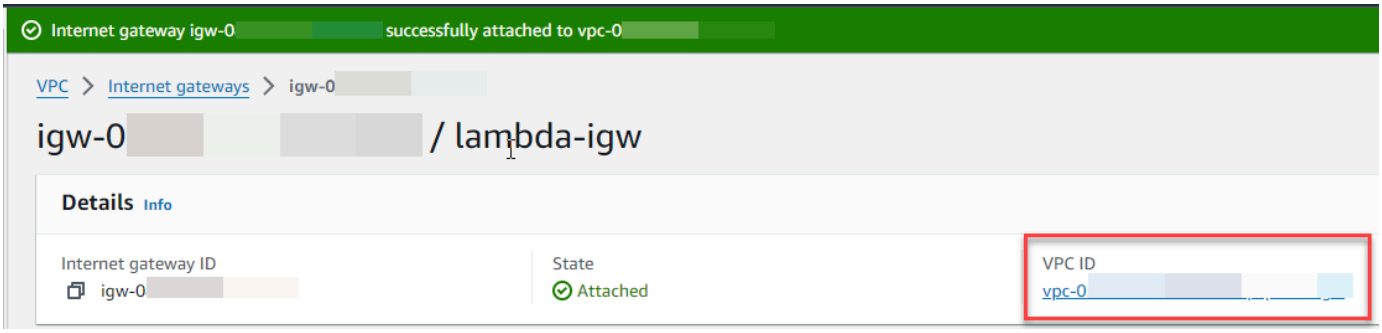
Ikuti langkah-langkah ini untuk membuat gateway internet, melampirkannya ke VPC Anda, dan menambahkannya ke tabel rute subnet publik Anda.

Untuk membuat gateway internet

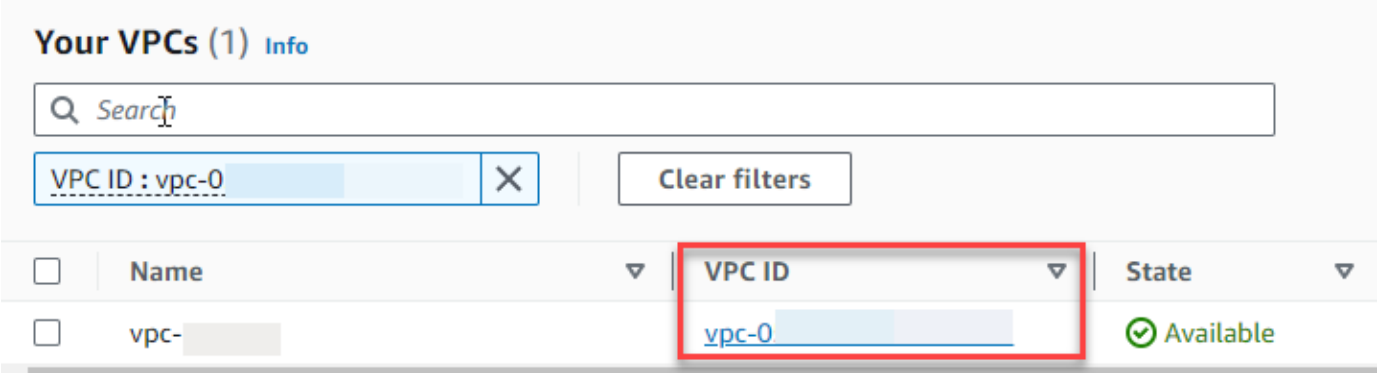
1. Buka konsol Amazon VPC di. https://console.aws.amazon.com/vpc/
2. Di panel navigasi, pilih gateway Internet.
3. Pilih Buat gateway internet.
4. (Opsional) Masukkan nama untuk gateway internet Anda.
5. (Opsional) Untuk menambahkan tag, pilih Tambahkan tag baru dan masukkan kunci tag dan nilai.
6. Pilih Buat gateway internet.
7. Pilih Lampirkan ke VPC dari spanduk di bagian atas layar, pilih VPC yang tersedia, lalu pilih Lampirkan gateway internet.



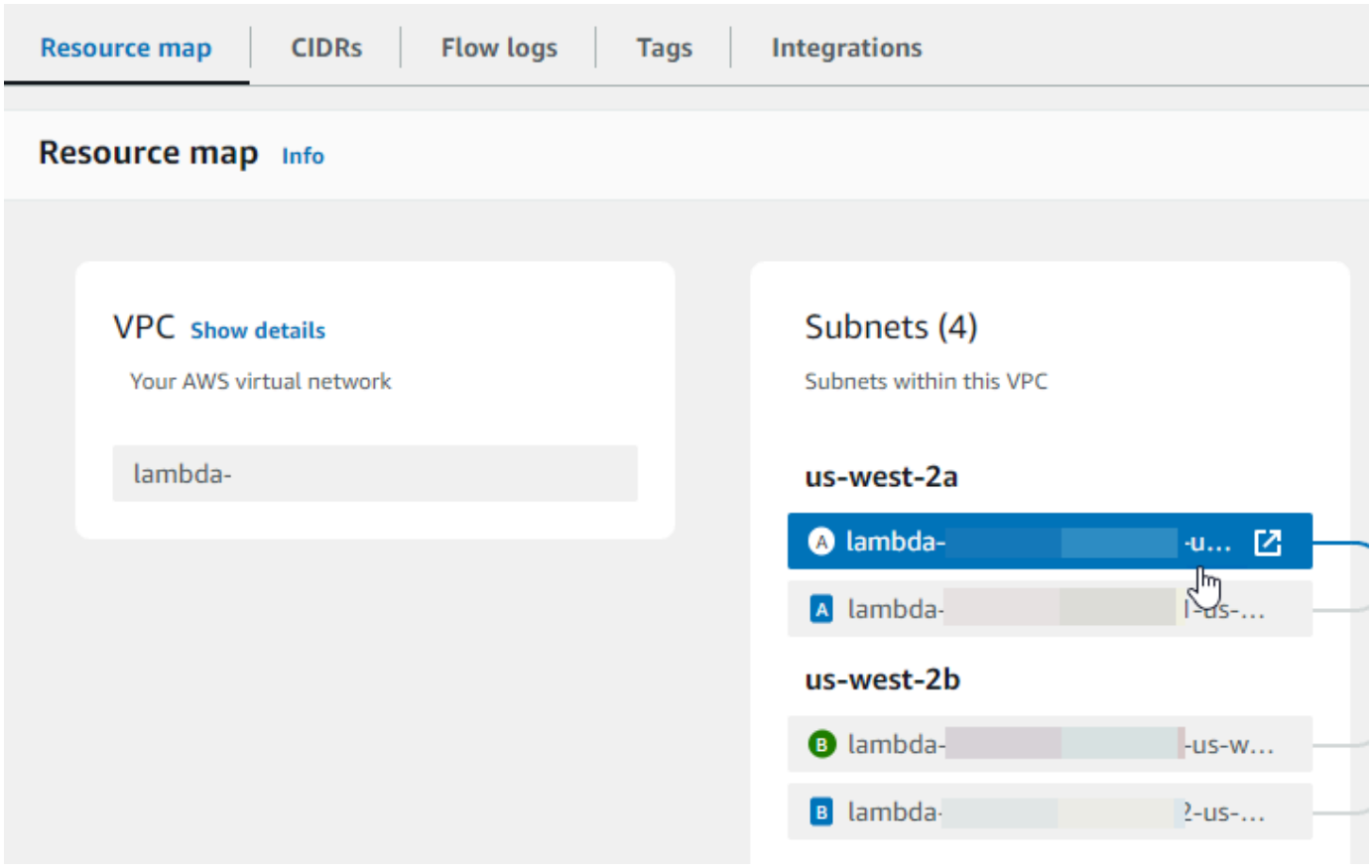
8. Pilih ID VPC.



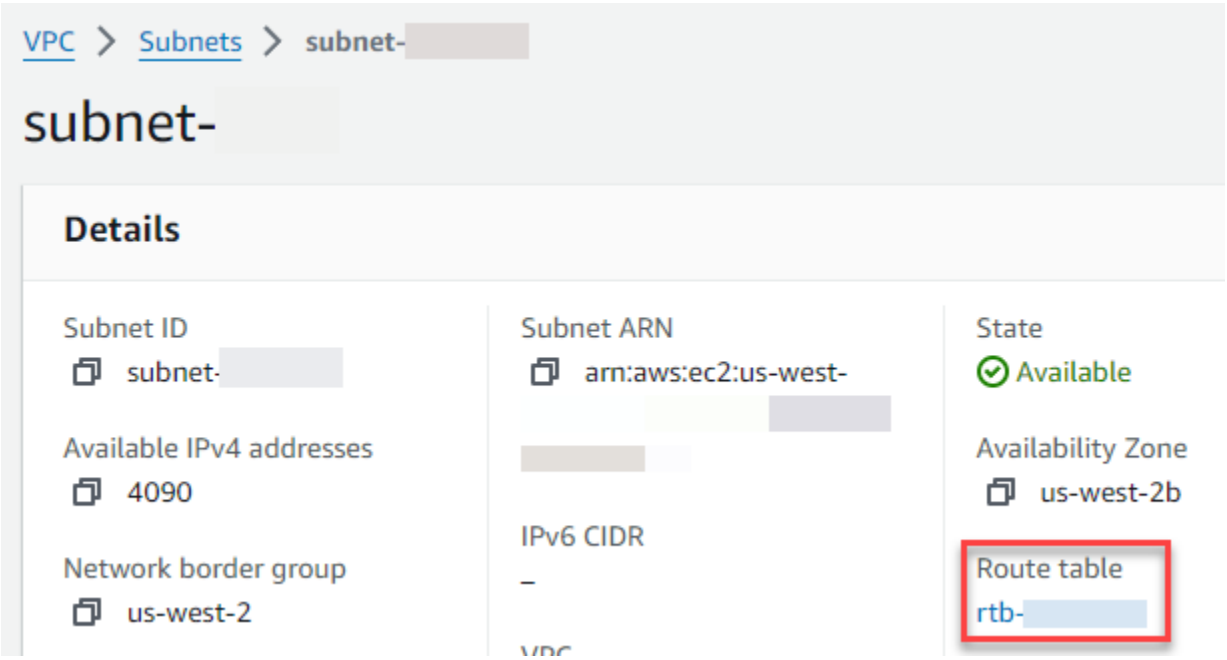
9. Pilih ID VPC lagi untuk membuka halaman detail VPC.



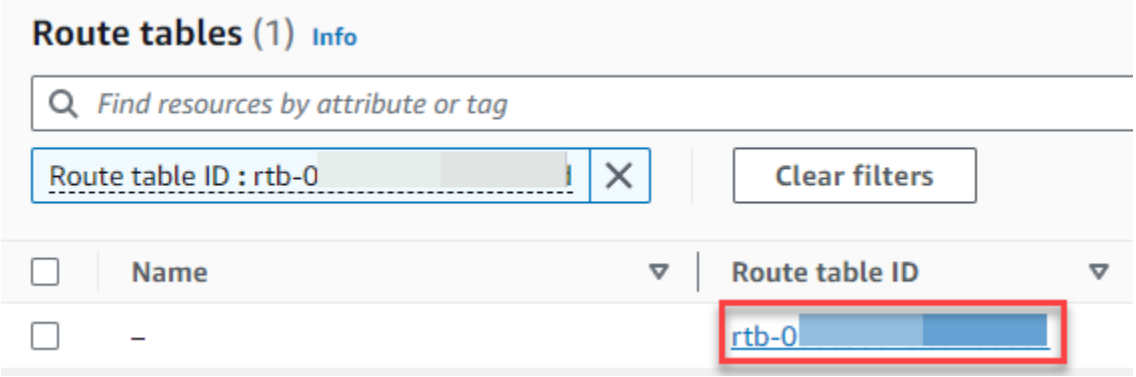
10. Gulir ke bawah ke bagian Peta sumber daya dan kemudian pilih subnet. Detail subnet ditampilkan di tab baru.



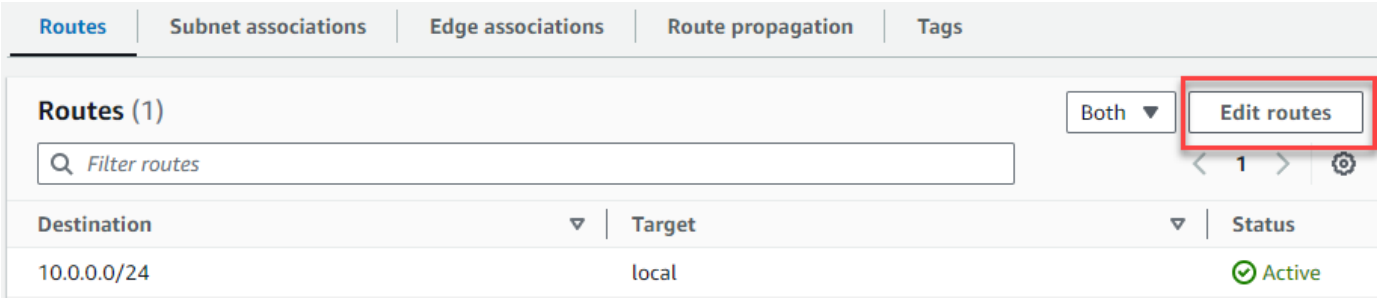
11. Pilih tautan di bawah Tabel rute.



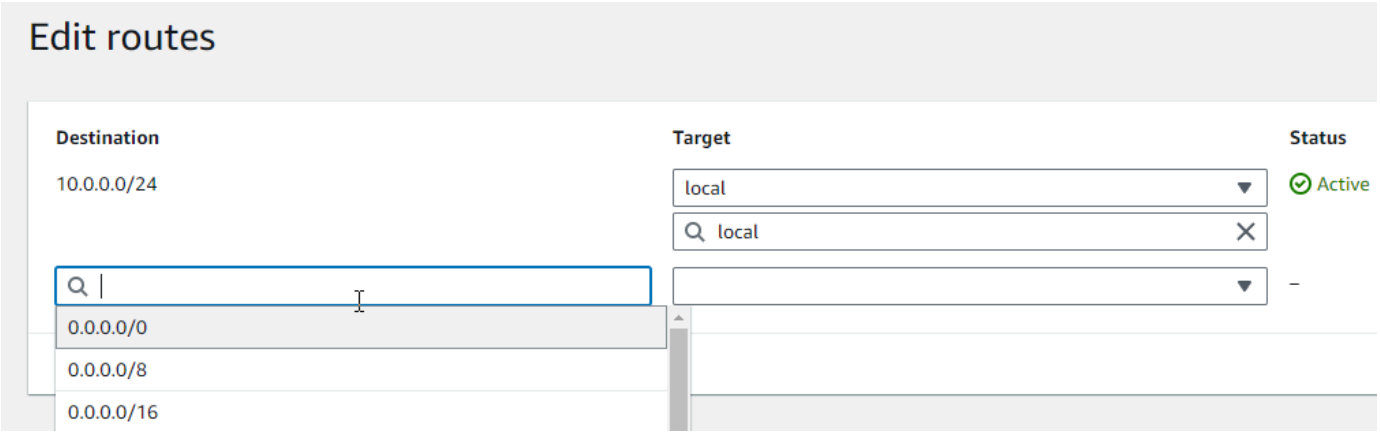
12. Pilih ID tabel Route untuk membuka halaman detail tabel rute.



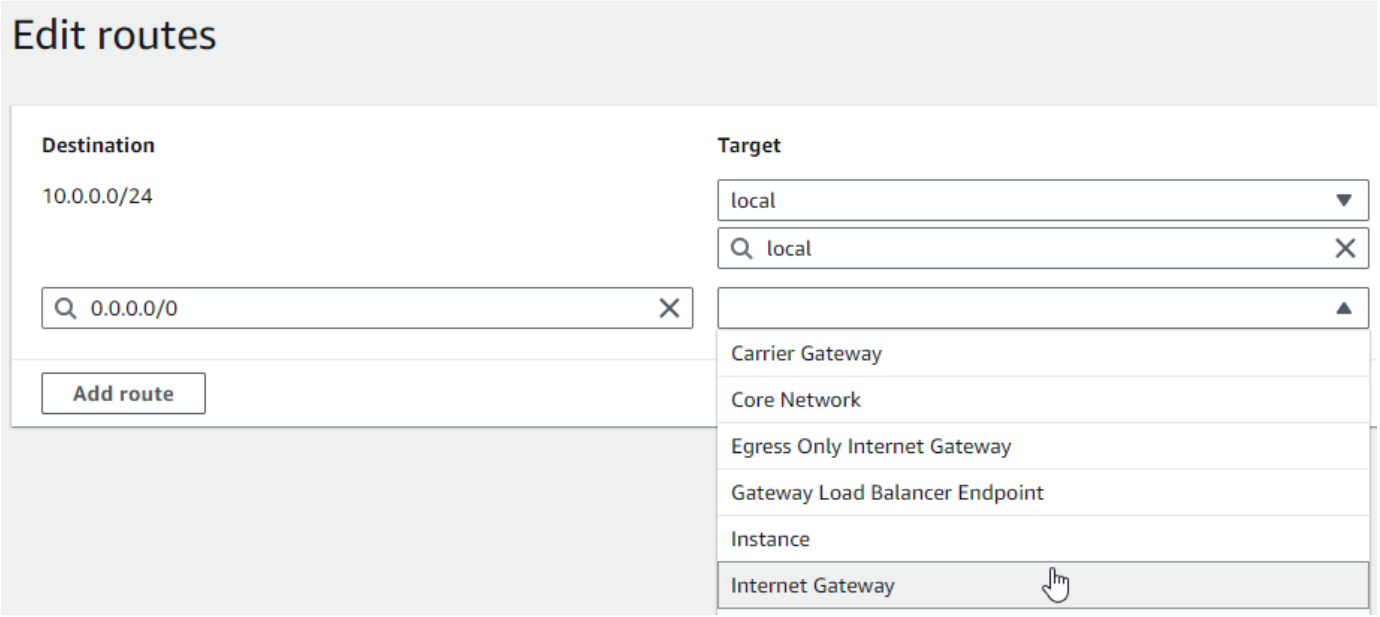
13. Di bawah Rute, pilih Edit rute.



14. Pilih Tambah rute, lalu masukkan `0.0.0.0/0` di kotak Tujuan.



15. Untuk Target, pilih Internet gateway, lalu pilih gateway internet yang Anda buat sebelumnya. Jika subnet Anda memiliki blok IPv6 CIDR, Anda juga harus menambahkan rute `::/0` ke gateway internet yang sama.



16. Pilih Simpan perubahan.

Buat gateway NAT

Ikuti langkah-langkah ini untuk membuat gateway NAT, mengaitkannya dengan subnet publik, dan kemudian menambahkannya ke tabel rute subnet pribadi Anda.

Untuk membuat gateway NAT dan mengaitkannya dengan subnet publik

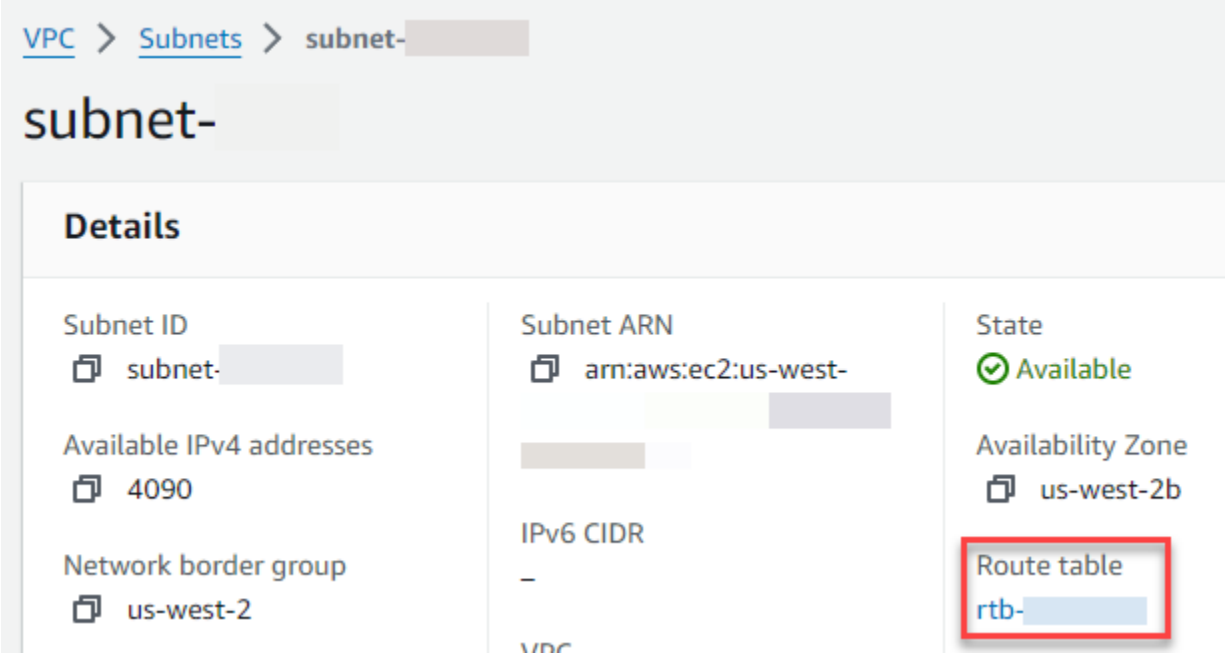
1. Di panel navigasi, pilih gateway NAT.
2. Pilih Buat gateway NAT.
3. (Opsional) Masukkan nama untuk gateway NAT Anda.
4. Untuk Subnet, pilih subnet publik di VPC Anda. (Subnet publik adalah subnet yang memiliki rute langsung ke gateway internet dalam tabel rutenya.)

> **Note**
>
> Gateway NAT dikaitkan dengan subnet publik, tetapi entri tabel rute ada di subnet pribadi.

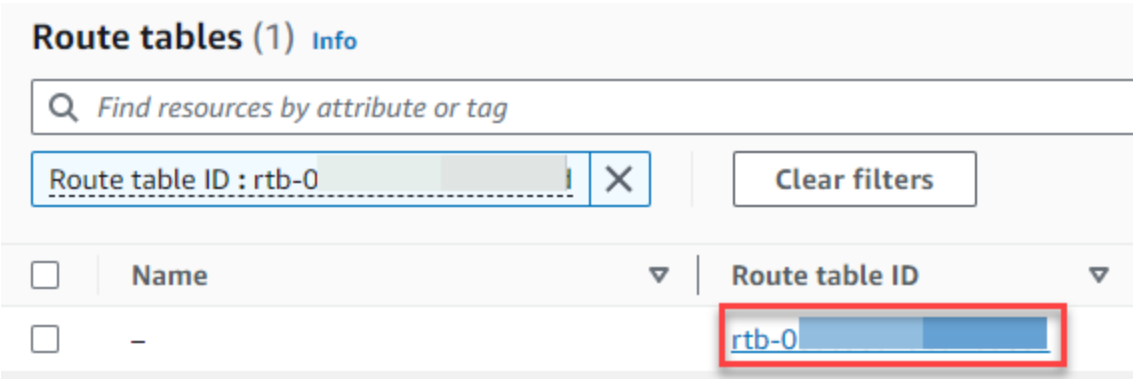
5. Untuk ID alokasi IP elastis, pilih alamat IP elastis atau pilih Alokasikan IP Elastis.
6. Pilih Buat gateway NAT.

Untuk menambahkan rute ke gateway NAT di tabel rute subnet pribadi

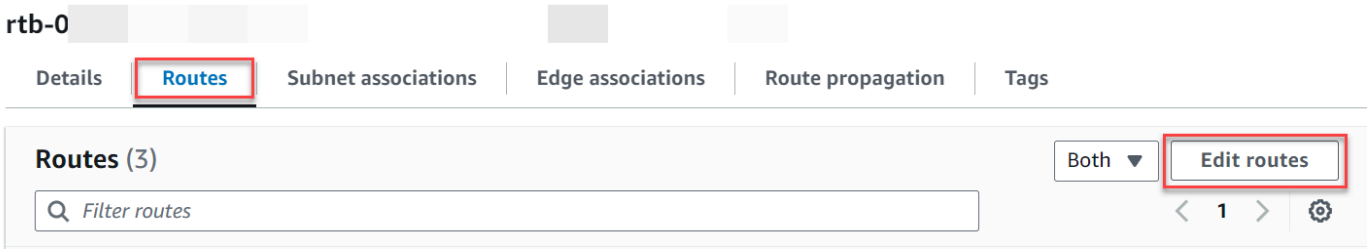
1. Di panel navigasi, pilih Pengguna.
2. Pilih subnet pribadi di VPC Anda. (Subnet pribadi adalah subnet yang tidak memiliki rute ke gateway internet di tabel rutenya.)
3. Pilih tautan di bawah Tabel rute.



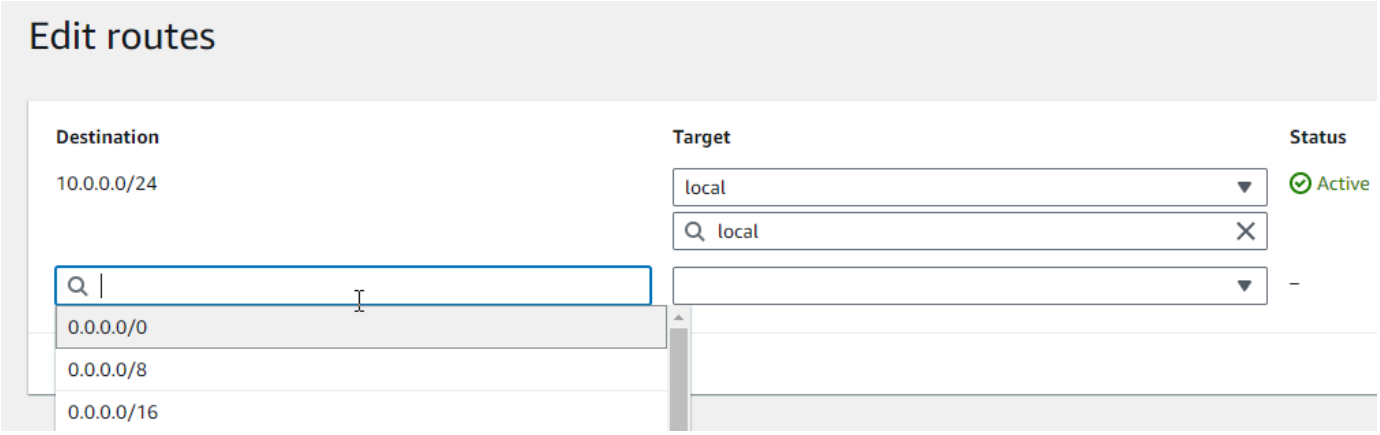
4. Pilih ID tabel Route untuk membuka halaman detail tabel rute.



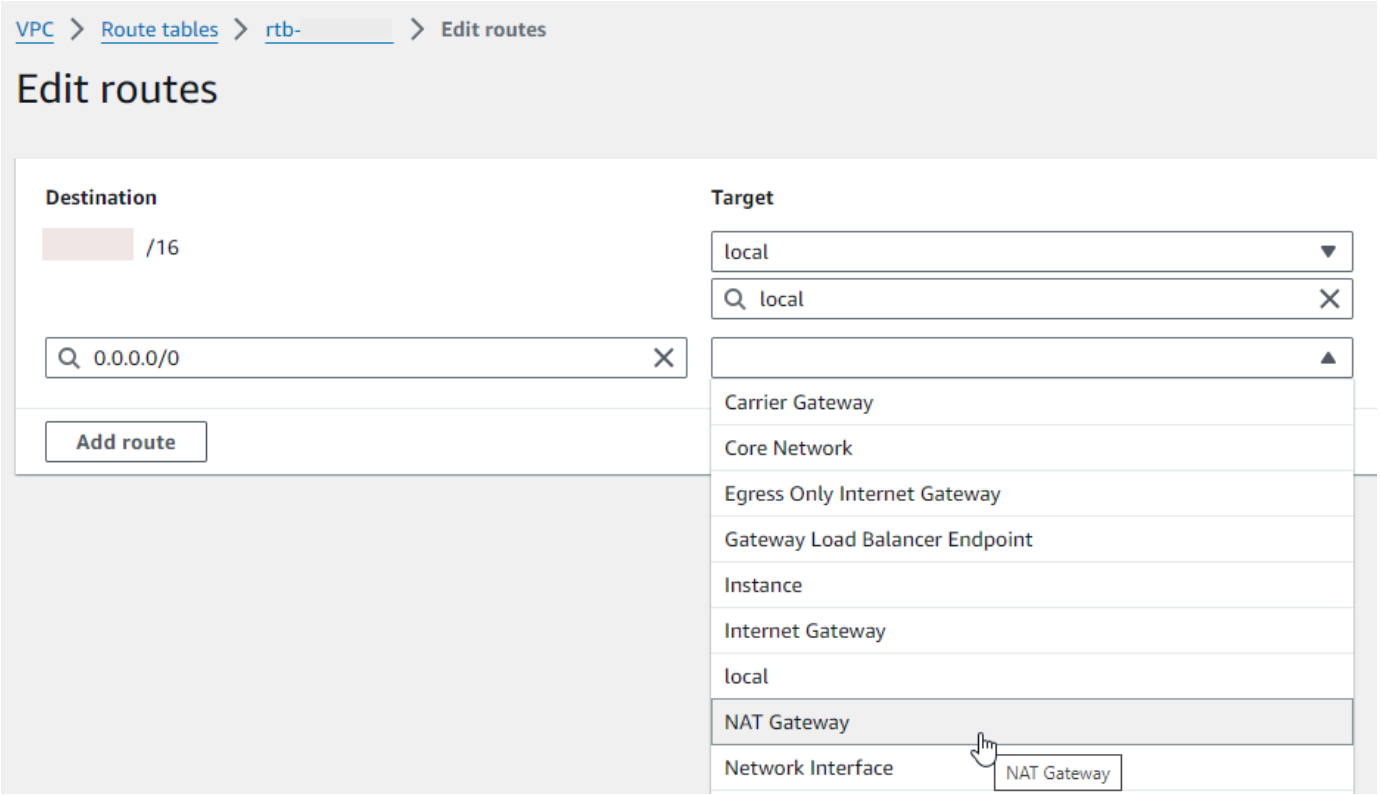
5. Gulir ke bawah dan pilih tab Rute, lalu pilih Edit rute



6. Pilih Tambah rute, lalu masukkan `0.0.0.0/0` di kotak Tujuan.



7. Untuk Target, pilih gateway NAT, lalu pilih gateway NAT yang Anda buat sebelumnya.



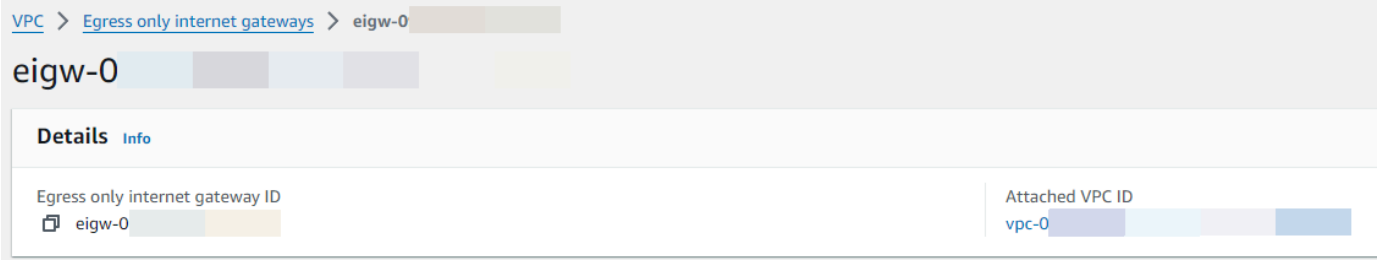
8. Pilih Simpan perubahan.

Buat gateway internet khusus egres (hanya IPv6)

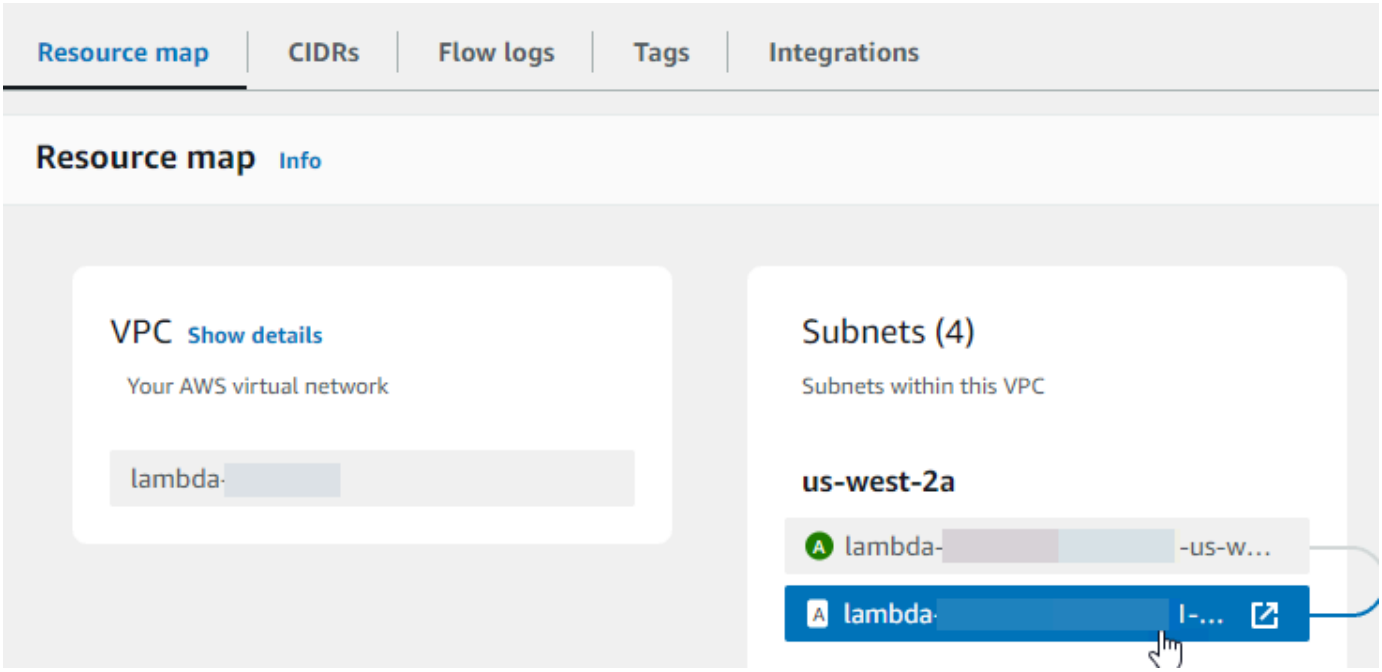
Ikuti langkah-langkah ini untuk membuat gateway internet khusus egres dan menambahkannya ke tabel rute subnet pribadi Anda.

Untuk membuat gateway internet egress-only

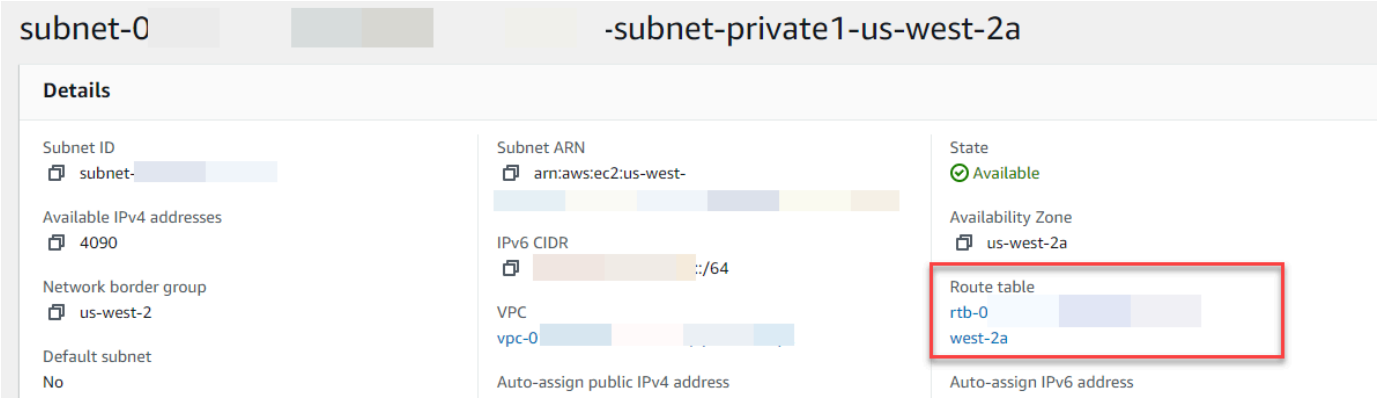
1. Di panel navigasi, pilih gateway Egress-only internet.
2. Pilih Buat gateway internet hanya jalan keluar.
3. (Opsional) Masukkan nama.
4. Pilih VPC tempat untuk membuat gateway internet egress-only.
5. Pilih Buat gateway internet hanya jalan keluar.
6. Pilih tautan di bawah ID VPC Terlampir.



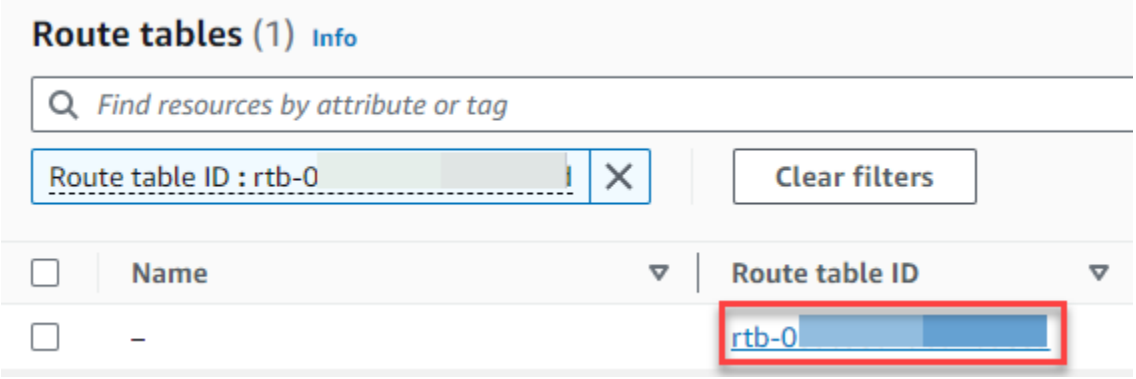
7. Pilih tautan di bawah ID VPC untuk membuka halaman detail VPC.
8. Gulir ke bawah ke bagian Peta sumber daya dan kemudian pilih subnet pribadi. (Subnet pribadi adalah subnet yang tidak memiliki rute ke gateway internet di tabel rutenya.) Detail subnet ditampilkan di tab baru.



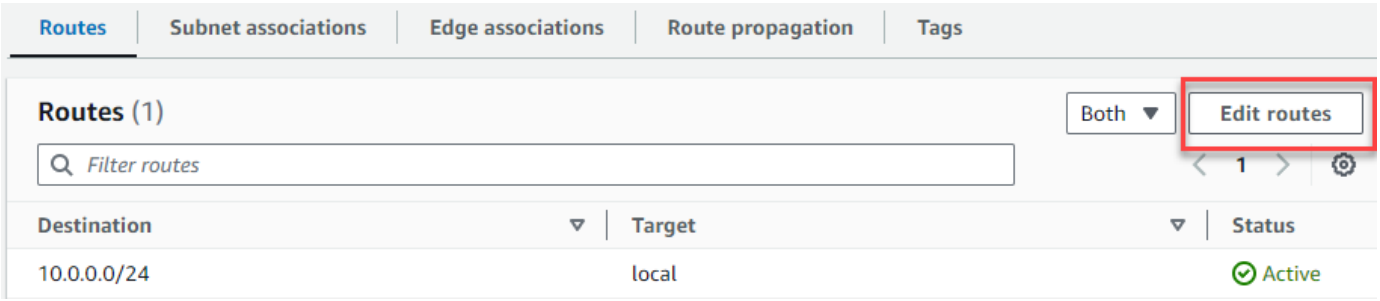
9. Pilih tautan di bawah Tabel rute.



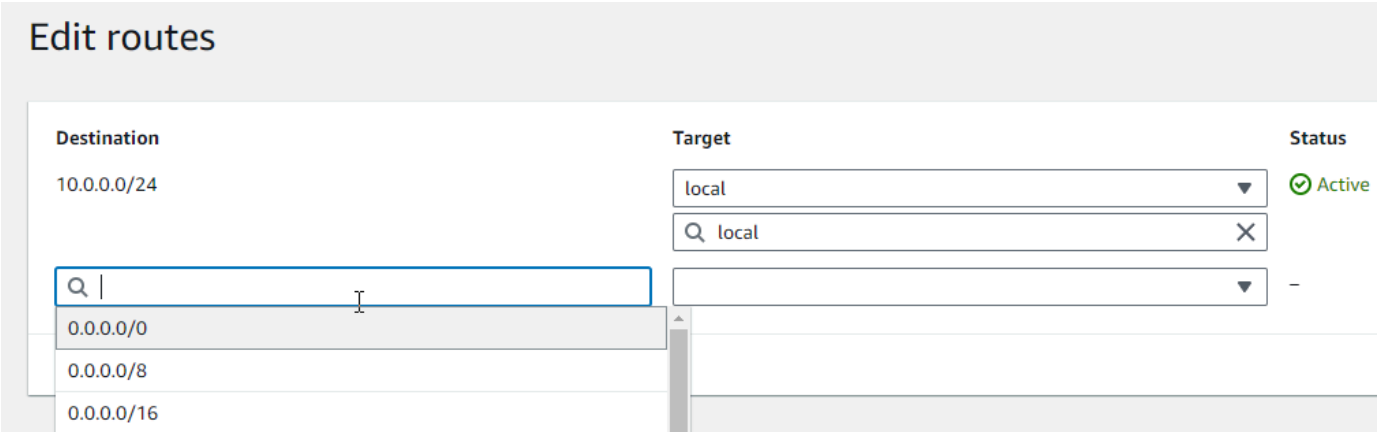
10. Pilih ID tabel Route untuk membuka halaman detail tabel rute.



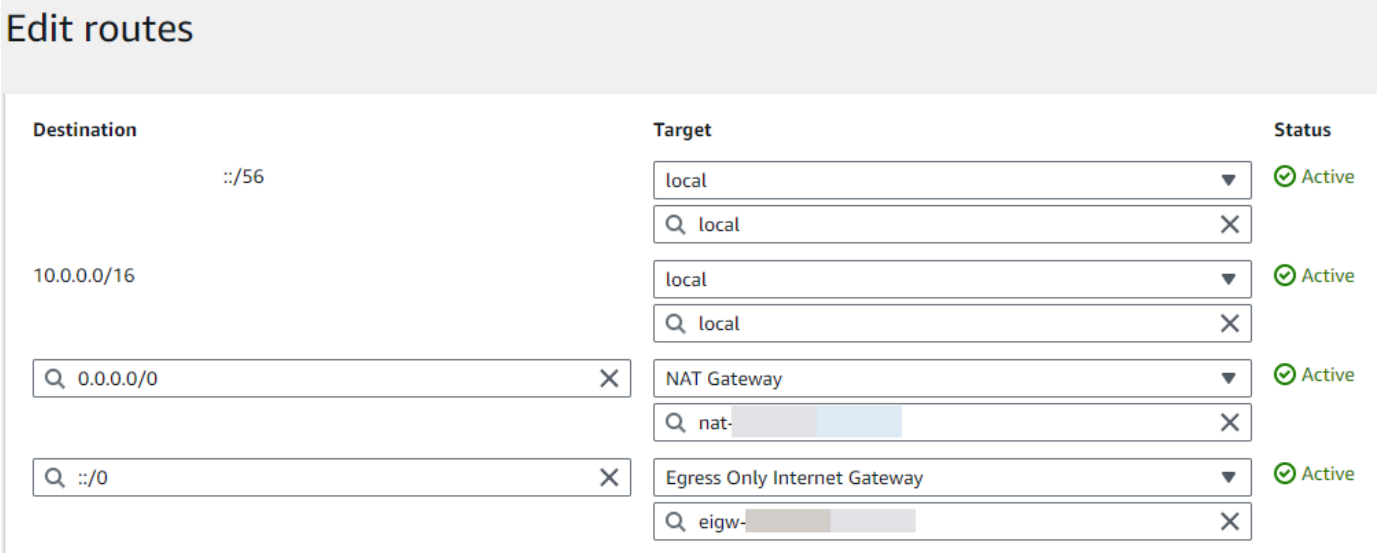
11. Di bawah Rute, pilih Edit rute.



12. Pilih Tambah rute, lalu masukkan `::/0` di kotak Tujuan.



13. Untuk Target, pilih Egress Only Internet Gateway, lalu pilih gateway yang Anda buat sebelumnya.



14. Pilih Simpan perubahan.

Konfigurasikan fungsi Lambda

Untuk mengonfigurasi VPC saat Anda membuat fungsi

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih Buat Fungsi Acara.
3. Di bawah Informasi Dasar, untuk Nama fungsi, masukkan nama untuk fungsi Anda.
4. Perluas Pengaturan lanjutan.
5. Pilih Aktifkan VPC, lalu pilih VPC.
6. (Opsional) Untuk mengizinkan lalu lintas IPv6 keluar, pilih Izinkan lalu lintas IPv6 untuk subnet dual-stack.
7. Untuk Subnet, pilih semua subnet pribadi. Subnet pribadi dapat mengakses internet melalui gateway NAT. Menghubungkan fungsi ke subnet publik tidak memberikan akses internet.

> **Note**
>
> Jika Anda memilih Izinkan lalu lintas IPv6 untuk subnet dual-stack, semua subnet yang dipilih harus memiliki blok CIDR IPv4 dan blok CIDR IPv6.

8. Untuk grup Keamanan, pilih grup keamanan yang memungkinkan lalu lintas keluar.

9. Pilih Buat Fungsi Acara.

Lambda secara otomatis membuat peran eksekusi dengan kebijakan AWSLambdaVPCAccessExecutionRole AWS terkelola. Izin dalam kebijakan ini hanya diperlukan untuk membuat antarmuka jaringan elastis untuk konfigurasi VPC, bukan untuk menjalankan fungsi Anda. Untuk menerapkan izin hak istimewa paling sedikit, Anda dapat menghapus AWSLambdaVPCAccessExecutionRolekebijakan dari peran eksekusi setelah membuat fungsi dan konfigurasi VPC. Untuk informasi selengkapnya, lihat Izin IAM yang diperlukan.

Untuk mengonfigurasi VPC untuk fungsi yang ada

Untuk menambahkan konfigurasi VPC ke fungsi yang ada, peran eksekusi fungsi harus memiliki izin untuk membuat dan mengelola antarmuka jaringan elastis. Kebijakan AWSLambdaVPCAccessExecutionRole AWS terkelola mencakup izin yang diperlukan. Untuk menerapkan izin hak istimewa paling sedikit, Anda dapat menghapus AWSLambdaVPCAccessExecutionRolekebijakan dari peran eksekusi setelah membuat konfigurasi VPC.

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih fungsi.
3. Pilih tab Konfigurasi, lalu pilih VPC.
4. Di bagian VPC, pilih Edit.
5. Pilih VPC.
6. (Opsional) Untuk mengizinkan lalu lintas IPv6 keluar, pilih Izinkan lalu lintas IPv6 untuk subnet dual-stack.
7. Untuk Subnet, pilih semua subnet pribadi. Subnet pribadi dapat mengakses internet melalui gateway NAT. Menghubungkan fungsi ke subnet publik tidak memberikan akses internet.

   > **Note**
   >
   > Jika Anda memilih Izinkan lalu lintas IPv6 untuk subnet dual-stack, semua subnet yang dipilih harus memiliki blok CIDR IPv4 dan blok CIDR IPv6.

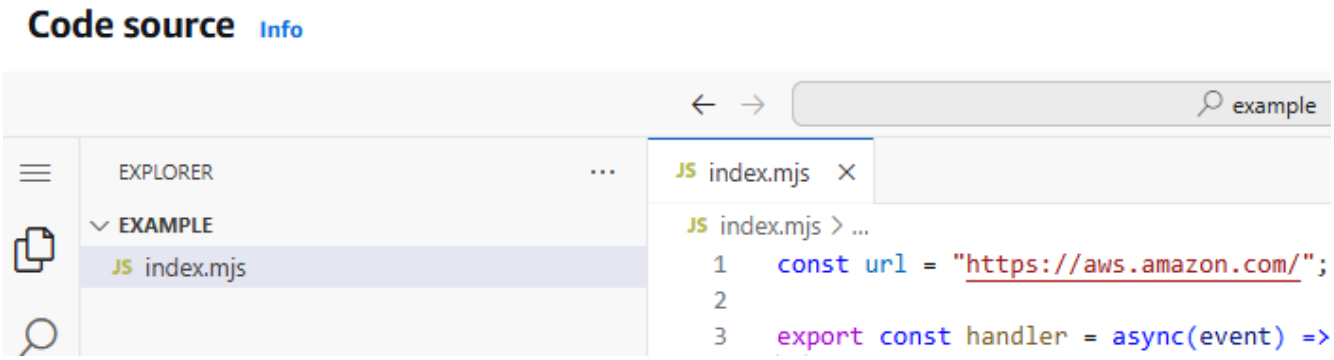
8. Untuk grup Keamanan, pilih grup keamanan yang memungkinkan lalu lintas keluar.
9. Pilih Simpan.

Uji fungsi

Gunakan kode contoh berikut untuk mengonfirmasi bahwa VPC-connected fungsi Anda dapat menjangkau internet publik. Jika berhasil, kode mengembalikan kode `200` status. Jika tidak berhasil, fungsi akan habis.

Node.js

1. Di panel Sumber kode di konsol Lambda, tempel kode berikut ke dalam file index.mjs. Fungsi ini membuat permintaan HTTP GET ke titik akhir publik dan mengembalikan kode respons HTTP untuk menguji apakah fungsi tersebut memiliki akses ke internet publik.

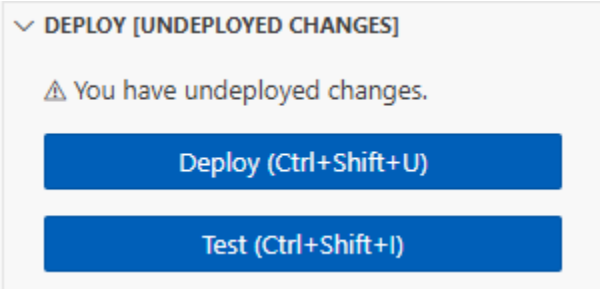


   Example— Permintaan HTTP dengan async/await

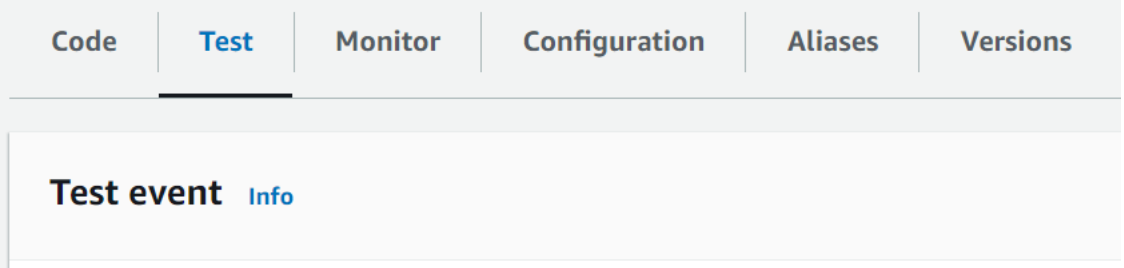
```
const url = "https://aws.amazon.com/";

export const handler = async(event) => {
    try {
        const res = await fetch(url);
        console.info("status", res.status);
        return res.status;
    }
    catch (e) {
        console.error(e);
        return 500;
    }
};
```

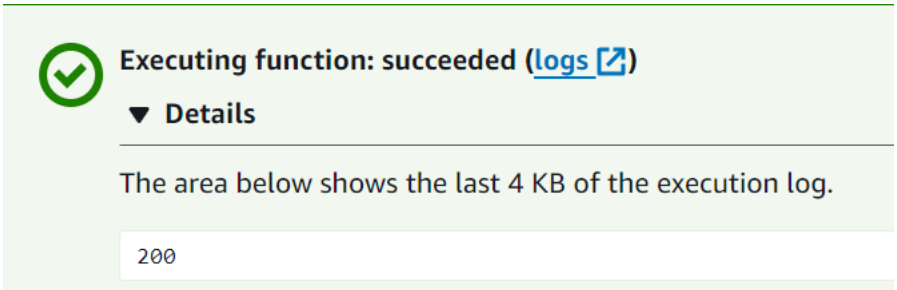
2. Di bagian DEPLOY, pilih Deploy untuk memperbarui kode fungsi Anda:



3. Pilih tab Uji.



4. Pilih Uji.
5. Fungsi mengembalikan kode `200` status. Ini berarti bahwa fungsi tersebut memiliki akses internet keluar.



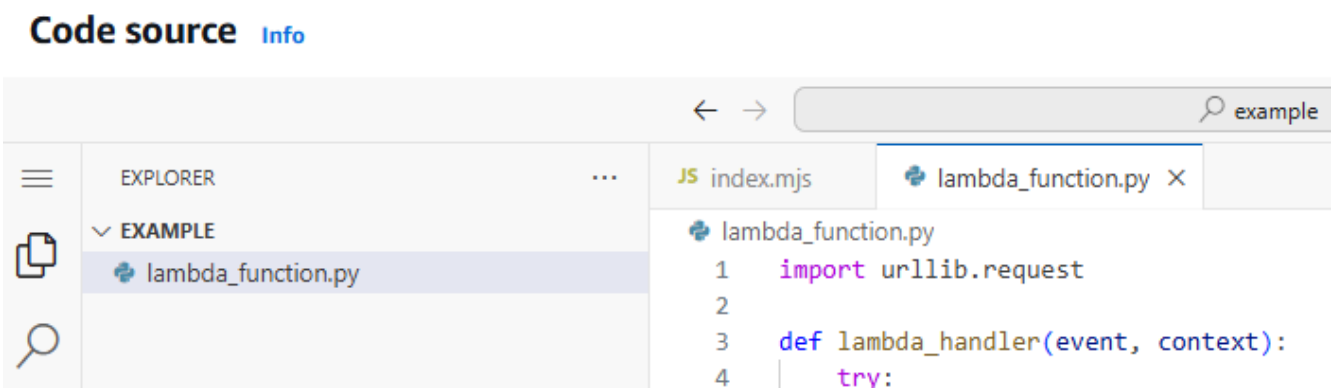
Jika fungsi tidak dapat menjangkau internet publik, Anda mendapatkan pesan kesalahan seperti ini:

```
{
  "errorMessage": "2024-04-11T17:22:20.857Z abe12jlc-640a-8157-0249-9be825c2y110
 Task timed out after 3.01 seconds"
}
```

Python

1. Di panel Sumber kode di konsol Lambda, tempel kode berikut ke dalam file lambda_function.py. Fungsi ini membuat permintaan HTTP GET ke titik akhir publik dan

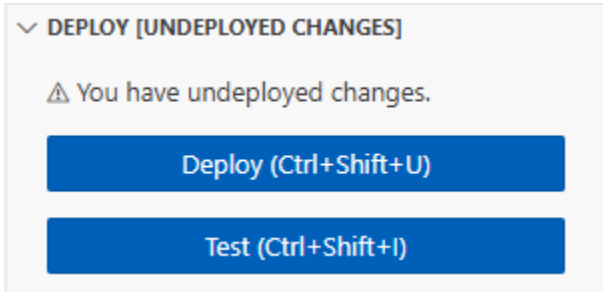
mengembalikan kode respons HTTP untuk menguji apakah fungsi tersebut memiliki akses ke internet publik.



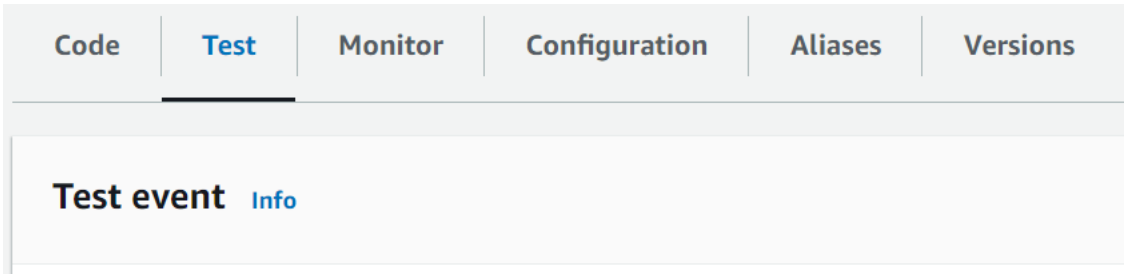

```
import urllib.request

def lambda_handler(event, context):
    try:
        response = urllib.request.urlopen('https://aws.amazon.com')
        status_code = response.getcode()
        print('Response Code:', status_code)
        return status_code
    except Exception as e:
        print('Error:', e)
        raise e
```

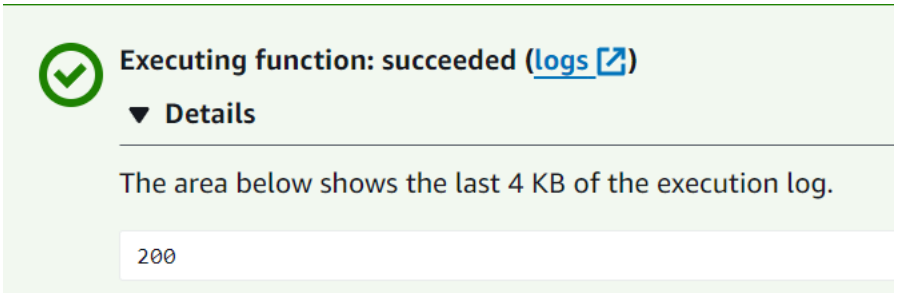
2. Di bagian DEPLOY, pilih Deploy untuk memperbarui kode fungsi Anda:



3. Pilih tab Uji.



4. Pilih Uji.
5. Fungsi mengembalikan kode `200` status. Ini berarti bahwa fungsi tersebut memiliki akses internet keluar.



Jika fungsi tidak dapat menjangkau internet publik, Anda mendapatkan pesan kesalahan seperti ini:

```
{
  "errorMessage": "2024-04-11T17:22:20.857Z abe12jlc-640a-8157-0249-9be825c2y110
 Task timed out after 3.01 seconds"
}
```

Menghubungkan titik akhir VPC antarmuka masuk untuk Lambda

Jika Anda menggunakan Amazon Virtual Private Cloud (Amazon VPC) untuk meng-host AWS sumber daya Anda, Anda dapat membuat koneksi antara VPC dan Lambda. Anda dapat menggunakan koneksi ini untuk mengaktifkan fungsi Lambda Anda tanpa menyeberangi internet publik.

Untuk membangun koneksi privat antara VPC Anda dan Lambda, buat VPC endpoint antarmuka. Titik akhir antarmuka didukung oleh AWS PrivateLink. Dengan AWS PrivateLink, Anda dapat mengakses API Lambda secara pribadi tanpa gateway internet, perangkat NAT, koneksi VPN, atau AWS Direct Connect koneksi. Instans dalam VPC Anda tidak memerlukan alamat IP publik untuk berkomunikasi dengan API Lambda. Lalu lintas antara VPC Anda dan Lambda tidak meninggalkan jaringan AWS .

Setiap titik akhir antarmuka diwakili oleh satu atau lebih antarmuka jaringan elastis dalam subnet Anda. Antarmuka jaringan menyediakan alamat IP privat yang berfungsi sebagai titik masuk untuk lalu lintas ke Lambda.

Bagian

- Pertimbangan untuk titik akhir antarmuka Lambda
- Membuat titik akhir antarmuka untuk Lambda
- Membuat kebijakan titik akhir antarmuka untuk Lambda

Pertimbangan untuk titik akhir antarmuka Lambda

Sebelum Anda mengatur titik akhir antarmuka untuk Lambda, pastikan untuk meninjau Properti dan batasan titik akhir antarmuka dalam Panduan Pengguna Amazon VPC.

Anda dapat memanggil salah satu operasi API Lambda dari VPC Anda. Misalnya, Anda dapat mengaktifkan fungsi Lambda dengan memanggil API `Invoke` dari VPC Anda. Untuk daftar lengkap API Lambda, lihat Tindakan dalam referensi API Lambda.

`use1-az3`adalah Wilayah kapasitas terbatas untuk fungsi Lambda VPC. Anda tidak boleh menggunakan subnet di zona ketersediaan ini dengan fungsi Lambda Anda karena hal ini dapat mengakibatkan pengurangan redundansi zonal jika terjadi pemadaman.

Keep-alive untuk koneksi persisten

Lambda menghapus koneksi idle dari waktu ke waktu, jadi Anda harus menggunakan arahan yang tetap aktif untuk mempertahankan koneksi yang persisten. Mencoba menggunakan kembali koneksi idle saat mengaktifkan fungsi akan menyebabkan kesalahan koneksi. Untuk mempertahankan koneksi yang persisten, gunakan arahan tetap aktif yang berkaitan dengan runtime Anda. Sebagai contoh, lihat Menggunakan Kembali Koneksi dengan Keep-Alive Node.js di Panduan Peng AWS SDK untuk JavaScript embang.

Pertimbangan Tagihan

Tidak ada biaya tambahan untuk mengakses fungsi Lambda melalui titik akhir antarmuka. Untuk informasi selengkapnya tentang harga Lambda, lihat AWS Lambda Harga.

Harga standar AWS PrivateLink berlaku untuk titik akhir antarmuka untuk Lambda. AWS Akun Anda ditagih untuk setiap jam titik akhir antarmuka disediakan di setiap Zona Ketersediaan dan untuk data yang diproses melalui titik akhir antarmuka. Untuk informasi selengkapnya tentang harga titik akhir antarmuka, lihat AWS PrivateLink harga.

Pertimbangan Peering VPC

Anda dapat menghubungkan VPC lain ke VPC dengan titik akhir antarmuka menggunakan peering VPC. Peering VPC adalah koneksi jaringan di antara dua VPC. Anda dapat menetapkan koneksi peering VPC di antara dua VPC milik Anda sendiri, atau dengan VPC di aku AWS lain. VPC juga dapat berada di dua Wil AWS ayah yang berbeda.

Lalu lintas antara VPC yang diintip tetap di AWS jaringan dan tidak melintasi internet publik. Setelah VPC diintip, sumber daya seperti instans Amazon Elastic Compute Cloud (Amazon EC2), instans Amazon Relational Database Service (Amazon RDS), atau fungsi VPC-enabled Lambda di kedua VPC dapat mengakses Lambda API melalui titik akhir antarmuka yang dibuat di salah satu VPC.

Membuat titik akhir antarmuka untuk Lambda

Anda dapat membuat titik akhir antarmuka untuk Lambda menggunakan konsol Amazon VPC atau AWS Command Line Interface ()AWS CLI. Untuk informasi selengkapnya, lihat Membuat titik akhir antarmuka dalam Panduan Pengguna Amazon VPC.

Untuk membuat titik akhir antarmuka untuk Lambda (konsol)

1. Buka Halaman titik akhir konsol Amazon VPC.

2. Pilih Buat Titik Akhir.
3. Untuk kategori Layanan, verifikasi Layanan AWS bahwa dipilih.
4. Untuk Nama Layanan, pilih com.amazonaws. ***region***.lambda. Verifikasi bahwa Jenis adalah Antarmuka.
5. Pilih VPC dan subnet.
6. Untuk mengaktifkan DNS privat untuk titik akhir antarmuka, pilih kotak centang Aktifkan Nama DNS. Sebaiknya aktifkan nama DNS pribadi untuk titik akhir VPC Anda untuk. Layanan AWS Ini memastikan bahwa permintaan yang menggunakan titik akhir layanan publik, seperti permintaan yang dibuat melalui AWS SDK, diselesaikan ke titik akhir VPC Anda.
7. Untuk Grup keamanan, pilih satu grup keamanan atau lebih.
8. Pilih Buat titik akhir.

Untuk menggunakan opsi DNS privat, Anda harus mengatur `enableDnsHostnames` dan `enableDnsSupportattributes` VPC Anda. Untuk informasi selengkapnya, lihat Melihat dan memperbarui dukungan DNS untuk VPC Anda di Panduan Pengguna Amazon VPC. Jika Anda mengaktifkan DNS privat untuk titik akhir antarmuka, Anda dapat membuat permintaan API untuk Lambda menggunakan nama DNS default untuk Wilayah, misalnya `lambda.us-east-1.amazonaws.com`. Untuk titik akhir layanan lainnya, lihat Titik akhir layanan dan kuota di. Referensi Umum AWS

Untuk informasi selengkapnya, lihat Mengakses layanan melalui titik akhir antarmuka dalam Panduan Pengguna Amazon VPC.

Untuk informasi tentang membuat dan mengonfigurasi titik akhir menggunakan CloudFormation, lihat sumber daya AWS: :EC2: :VPCendPoint di Panduan Pengguna. AWS CloudFormation

Untuk membuat titik akhir antarmuka untuk Lambda (AWS CLI)

Gunakan perintah create-vpc-endpoint dan tentukan ID VPC, tipe titik akhir VPC (antarmuka), nama layanan, subnet yang menggunakan titik akhir, dan grup keamanan untuk dikaitkan dengan antarmuka jaringan titik akhir. Contoh:

```
aws ec2 create-vpc-endpoint
  --vpc-id vpc-ec43eb89
  --vpc-endpoint-type Interface
  --service-name com.amazonaws.us-east-1.lambda
  --subnet-id subnet-abababab
```

```
  --security-group-id sg-1a2b3c4d
```

Membuat kebijakan titik akhir antarmuka untuk Lambda

Untuk mengontrol siapa yang dapat menggunakan titik akhir antarmuka Anda dan fungsi Lambda mana yang dapat diakses pengguna, Anda dapat melampirkan kebijakan titik akhir ke titik akhir Anda. Kebijakan menentukan informasi berikut ini:

- Prinsip yang dapat melakukan tindakan.
- Tindakan yang dapat dilakukan oleh prinsip.
- Sumber daya yang dapat digunakan untuk melakukan tindakan.

Untuk informasi selengkapnya, lihat Mengontrol akses ke layanan dengan VPC endpoint dalam Panduan Pengguna Amazon VPC.

Kebijakan titik akhir antarmuka untuk tindakan Lambda

Berikut adalah contoh kebijakan titik akhir untuk Lambda. Jika dilampirkan ke titik akhir, kebijakan ini memungkinkan `MyUser` pengguna untuk mengaktifkan fungsi `my-function`.

> **Note**
>
> Anda perlu memasukkan fungsi ARN yang memenuhi syarat dan tidak memenuhi syarat dalam sumber daya.

```
{
   "Statement":[
      {
         "Principal":
         {
             "AWS": "arn:aws:iam::111122223333:user/MyUser"
         },
         "Effect":"Allow",
         "Action":[
            "lambda:InvokeFunction"
         ],
         "Resource": [
               "arn:aws:lambda:us-east-2:123456789012:function:my-function",
```

```
                "arn:aws:lambda:us-east-2:123456789012:function:my-function:*"
            ]
        }
    ]
}
```

Untuk memberikan fungsi Anda akses ke sumber daya di VPC milik VPC lain Akun AWS, lihatthe section called “Melampirkan fungsi ke sumber daya di akun lain”.

Mengkonfigurasi akses sistem file untuk Lambda Event Functions

Anda dapat mengonfigurasi fungsi Lambda untuk memasang sistem file ke direktori lokal. Lambda mendukung jenis-jenis sistem file berikut:

- Amazon Elastic File System (Amazon EFS) — Sistem file tanpa server yang menskalakan secara otomatis dengan beban kerja Anda.
- File Amazon S3 — Sistem file tanpa server untuk memasang bucket Amazon S3 Anda. File Amazon S3 menyediakan akses ke objek Amazon S3 Anda sebagai file yang menggunakan operasi sistem file standar seperti baca dan tulis di jalur pemasangan lokal.

> **Note**
>
> Fungsi Lambda dapat menggunakan File Amazon EFS atau Amazon S3, tetapi tidak keduanya. Jika fungsi Anda sudah dikonfigurasi dengan satu jenis sistem file, Anda harus menghapusnya sebelum mengonfigurasi yang lain.

Mengonfigurasi akses sistem file Amazon EFS

Anda dapat mengonfigurasikan fungsi untuk memasang sistem file Amazon Elastic File System (Amazon EFS) ke direktori lokal. Amazon EFS adalah sistem file tanpa server yang menskalakan secara otomatis dengan beban kerja Anda. Dengan Amazon EFS, kode fungsi Anda dapat mengakses dan memodifikasi sumber daya bersama dengan aman dan dengan konkurensi tinggi.

Bagian-bagian

- Peran eksekusi dan izin pengguna
- Mengonfigurasi sistem file dan titik akses
- Menyambung ke sistem file (konsol)
- Wilayah yang Didukung

Peran eksekusi dan izin pengguna

Jika sistem file tidak memiliki kebijakan yang dikonfigurasi pengguna AWS Identity and Access Management (IAM), EFS menggunakan kebijakan default yang memberikan akses penuh ke klien mana pun yang dapat terhubung ke sistem file menggunakan target pemasangan sistem file. Jika

sistem file memiliki kebijakan IAM yang dikonfigurasi pengguna, peran eksekusi fungsi Anda harus memiliki izin yang benar. `elasticfilesystem`

Izin peran eksekusi

- sistem file elastis: ClientMount
- elasticfilesystem: ClientWrite (tidak diperlukan untuk koneksi hanya-baca)

Izin ini disertakan dalam kebijakan terkelola AmazonElasticFileSystemClientReadWriteAccess. Selain itu, peran eksekusi Anda harus memiliki izin yang diperlukan untuk terhubung ke VPC sistem file.

Saat Anda mengonfigurasi sistem file, Lambda menggunakan izin Anda untuk memverifikasi target pemasangan. Untuk mengonfigurasi fungsi untuk terhubung ke sistem file, pengguna Anda memerlukan izin berikut:

Izin pengguna

- sistem file elastis: DescribeMountTargets

Mengonfigurasi sistem file dan titik akses

Buat sistem file di Amazon EFS dengan target pemasangan di setiap Availability Zone yang tehubung dengan fungsi Anda. Untuk performa dan ketahanan, gunakan setidaknya dua Availability Zone. Misalnya, dalam konfigurasi sederhana, Anda dapat memiliki VPC dengan dua subnet privat di Availability Zone yang berbeda. Fungsi ini terhubung ke subnet dan target pemasangan tersedia pada masing-masing subnet. Pastikan lalu lintas NFS (port 2049) diizinkan oleh grup keamanan yang digunakan oleh fungsi dan target pemasangan.

> **Note**
>
> Saat Anda membuat sistem file, Anda memilih mode performa yang tidak dapat diubah lagi nantinya. Mode tujuan umum memiliki latensi yang lebih rendah, dan I/O mode Max mendukung throughput maksimum dan IOPS yang lebih tinggi. Untuk bantuan saat memilih, lihat Performa Amazon EFS dalam Panduan Pengguna Amazon Elastic File System.

Titik akses menghubungkan setiap instans fungsi ke target pemasangan yang tepat untuk Availability Zone yang terhubung dengannya. Untuk performa terbaik, buat titik akses dengan alur non-akar, dan batasi jumlah file yang Anda buat di setiap direktori. Contoh berikut membuat direktori bernama `my-`

`function` pada sistem file dan menetapkan ID pemilik menjadi 1001 dengan izin direktori standar (755).

Example konfigurasi titik akses

- Nama – `files`
- ID Pengguna – `1001`
- ID Grup – `1001`
- Jalur – `/my-function`
- Izin – `755`
- ID pengguna pemilik – `1001`
- ID pengguna grup – `1001`

Saat fungsi menggunakan titik akses, fungsi diberikan ID pengguna 1001 dan memiliki akses penuh ke direktori.

Untuk informasi selengkapnya, lihat topik berikut di Panduan Pengguna Amazon Elastic File System:

- Membuat sumber daya untuk Amazon EFS
- Bekerja dengan pengguna, grup, dan izin

Menyambung ke sistem file (konsol)

Fungsi terhubung ke sistem file melalui jaringan lokal di VPC. Subnet yang terhubung ke fungsi Anda dapat berupa subnet yang sama yang berisi titik pemasangan untuk sistem file Anda, atau subnet di Availability Zone yang sama yang dapat mengarahkan lalu lintas NFS (port 2049) ke sistem file.

> **Note**
>
> Jika fungsi Anda belum terhubung ke VPC, lihat Memberikan Lambda Event Functions akses ke sumber daya di Amazon VPC.

Untuk mengkonfigurasi akses sistem file EFS

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih fungsi.

3. Pilih Konfigurasi, lalu pilihSistem file.
4. Di bagian Sistem file, pilih Tambahkan sistem file.
5. Pilih EFS.
6. Konfigurasikan properti berikut:

   - Sistem file EFS – Titik akses untuk sistem file dalam VPC yang sama.
   - Jalur pemasangan lokal – Lokasi tempat sistem file dipasang pada fungsi Lambda, memulai dengan `/mnt/`.

> **Harga**
>
> Amazon EFS mengenakan biaya untuk penyimpanan dan throughput, dengan harga yang berbeda-beda menurut kelas penyimpanan. Untuk detailnya, lihat Harga Amazon EFS. Lambda mengenakan biaya untuk transfer data di antara VPC. Ini hanya berlaku jika VPC fungsi Anda dihubungkan ke VPC lain dengan sistem file. Harganya sama dengan transfer data Amazon EC2 di antara VPC pada Wilayah yang sama. Untuk detailnya, lihat Harga Lambda.

Wilayah yang Didukung

Amazon EFS untuk Lambda tersedia di semua Wilayah komersial kecuali Asia Pasifik (Selandia Baru), Asia Pasifik (Taipei), Asia Pasifik (Malaysia), Asia Pasifik (Thailand), dan Kanada Barat (Calgary).

Mengkonfigurasi akses File Amazon S3

File Amazon S3 menghadirkan sistem file bersama yang menghubungkan sumber daya AWS komputasi apa pun secara langsung dengan data Anda di Amazon S3. File Amazon S3 menyediakan akses ke objek Amazon S3 Anda sebagai file yang menggunakan operasi sistem file standar seperti baca dan tulis di jalur pemasangan lokal. Pelajari selengkapnya tentang File Amazon S3.

Bagian-bagian

- Prasyarat dan pengaturan
- Peran eksekusi dan izin pengguna
- Menyambung ke sistem file (konsol)
- Mengkonfigurasi pembacaan langsung

Prasyarat dan pengaturan

Sebelum Anda mengatur File Amazon S3 dengan fungsi Lambda Anda, pastikan Anda memiliki yang berikut:

- Sistem file Amazon S3 dan memasang target dalam status yang tersedia di akun yang sama dan Wilayah AWS sebagai fungsi Lambda Anda.
- Fungsi Lambda dalam VPC yang sama dengan target pemasangan. Anda harus memiliki target mount di setiap subnet tempat fungsi Anda diterapkan.
- Grup keamanan yang memungkinkan lalu lintas NFS (port 2049) antara fungsi Lambda Anda dan target pemasangan. Pelajari lebih lanjut tentang mengonfigurasi grup keamanan.

Untuk informasi selengkapnya, lihat topik berikut di Panduan Pengguna Amazon S3:

- Memulai dengan File Amazon S3
- Prasyarat File Amazon S3
- Praktik terbaik File Amazon S3

Peran eksekusi dan izin pengguna

Peran eksekusi fungsi Anda harus memiliki izin berikut untuk mengakses sistem file File Amazon S3:

Izin peran eksekusi

- s3files: ClientMount - Diperlukan untuk me-mount sistem file.
- s3files: ClientWrite - Diperlukan untuk akses baca-tulis. Tidak diperlukan untuk koneksi read-only.

Izin ini disertakan dalam kebijakan terkelola AmazonS3FilesClientReadWriteAccess. Selain itu, peran eksekusi Anda harus memiliki izin yang diperlukan untuk terhubung ke VPC sistem file.

> **Note**
>
> File Amazon S3 mengoptimalkan throughput dengan mengalirkan permintaan baca yang memenuhi syarat langsung dari bucket Amazon S3 Anda. Pembacaan langsung diaktifkan secara default untuk fungsi yang dikonfigurasi dengan memori 512 MB atau lebih. Untuk informasi selengkapnya, lihat Mengkonfigurasi pembacaan langsung.

Fungsi Anda juga memerlukan izin berikut untuk membaca langsung dari Amazon S3:

- s3: GetObject
- s3: GetObjectVersion

Untuk informasi selengkapnya tentang izin yang diperlukan, lihat izin IAM untuk File Amazon S3 di Panduan Pengguna Amazon S3.

Saat Anda mengonfigurasi sistem file di konsol, Lambda menggunakan izin Anda untuk memverifikasi target pemasangan dan titik akses. Untuk mengonfigurasi fungsi untuk terhubung ke sistem file, pengguna Anda memerlukan izin berikut:

Izin pengguna

- s3file: ListFileSystems
- s3file: ListAccessPoints
- s3file: GetFileSystem
- s3file: GetAccessPoint
- s3files: CreateAccessPoint — Diperlukan jika melampirkan sistem file ke fungsi dari konsol.

Contoh kebijakan berikut memberikan izin peran eksekusi fungsi Anda untuk memasang sistem file Amazon S3 dengan akses baca-tulis dan baca langsung dari Amazon S3.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "S3FilesLambdaAccess",
            "Effect": "Allow",
            "Action": [
                "s3files:ClientMount",
                "s3files:ClientWrite"
            ],
            "Resource": "*"
        },
        {
            "Sid": "S3DirectRead",
            "Effect": "Allow",
            "Action": [
```

```
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": "arn:aws:s3:::bucket-name/*"
        },
        {
            "Sid": "S3FilesConsoleSetup",
            "Effect": "Allow",
            "Action": [
                "s3files:ListFileSystems",
                "s3files:ListAccessPoints",
                "s3files:GetFileSystem",
                "s3files:GetAccessPoint",
                "s3files:CreateAccessPoint"
            ],
            "Resource": "*"
        }
    ]
}
```

Menyambung ke sistem file (konsol)

Fungsi terhubung ke sistem file melalui jaringan lokal di VPC. Subnet yang terhubung ke fungsi Anda dapat berupa subnet yang sama yang berisi titik pemasangan untuk sistem file Anda, atau subnet di Availability Zone yang sama yang dapat mengarahkan lalu lintas NFS (port 2049) ke sistem file.

> **Note**
>
> Jika fungsi Anda belum terhubung ke VPC, lihat Memberikan Lambda Event Functions akses ke sumber daya di Amazon VPC.

Untuk mengkonfigurasi akses File S3

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih fungsi.
3. Pilih Konfigurasi, lalu pilih Sistem file.
4. Pilih Tambahkan sistem file (atau Edit untuk mengubah konfigurasi yang ada).
5. Pilih File S3.
6. Konfigurasikan properti berikut:

- Sistem file S3 - Pilih sistem file dari dropdown.
- Titik akses (opsional) - Pilih titik akses. Jika sistem file tidak memiliki titik akses, Lambda secara otomatis membuat satu ketika Anda menyimpan (UID/GID 1000:1000, direktori `/lambda` root, izin 755). Jika titik akses ada, Anda harus memilih satu.
- Jalur pemasangan lokal – Lokasi tempat sistem file dipasang pada fungsi Lambda, memulai dengan `/mnt/`.
- Konfigurasi lanjutan
  - Pembacaan S3 Langsung — Mengontrol apakah Lambda dapat melakukan streaming pembacaan yang memenuhi syarat langsung dari bucket Amazon S3 Anda untuk throughput yang lebih tinggi. Untuk informasi selengkapnya, lihat Mengkonfigurasi pembacaan langsung.

7. Pilih Simpan.

Sistem file Anda dilampirkan saat berikutnya Anda menjalankan fungsi Lambda Anda.

Mengkonfigurasi pembacaan langsung

Pembacaan langsung mengoptimalkan throughput dengan streaming data langsung dari bucket Amazon S3 Anda. Lambda menggunakan pembacaan langsung untuk pembacaan besar dan ketika penyimpanan kinerja tinggi sistem file tidak menyimpan data file. Untuk informasi selengkapnya, lihat Cara File S3 memberikan kinerja. Konfigurasi opsional ini adalah bagian `FileSystemConfigs` dari fungsi Anda dan hanya berlaku untuk sistem file Amazon S3. `DirectS3Read`Pengaturan menerima nilai-nilai berikut:

- `AUTO`(default) - Memungkinkan pembacaan langsung untuk fungsi yang dikonfigurasi dengan memori 512 MB atau lebih.
- `ENABLED`— Memungkinkan pembacaan langsung untuk fungsi Anda, termasuk fungsi dengan memori kurang dari 512 MB.
- `DISABLED`— Rute semua membaca melalui penyimpanan kinerja tinggi sistem file.

> **Note**
>
> Jika pembacaan langsung gagal, Lambda secara otomatis kembali membaca melalui sistem file.

Untuk menggunakan pembacaan langsung, peran eksekusi fungsi Anda harus memiliki izin yang diperlukan. Untuk informasi selengkapnya, lihat kebijakan prasyarat File Amazon S3.

Buat alias untuk Fungsi Acara Lambda

Anda dapat membuat alias untuk fungsi Lambda Anda. Alias Lambda adalah penunjuk ke versi fungsi yang dapat Anda perbarui. Pengguna fungsi dapat mengakses versi fungsi menggunakan alias Amazon Resource Name (ARN). Saat menerapkan versi baru, Anda dapat memperbarui alias untuk menggunakan versi baru, atau membagi lalu lintas di antara dua versi. Untuk informasi selengkapnya tentang penggunaan alias dalam aplikasi Anda, lihatthe section called “Menggunakan alias”.

Console

Untuk membuat alias menggunakan konsol

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih fungsi.
3. Pilih Alias, lalu pilih Buat alias.
4. Di halaman Buat alias, lakukan hal berikut:
   a. Masukkan Nama untuk alias.
   b. (Opsional) Masukkan Deskripsi untuk alias.
   c. Untuk Versi, pilih versi fungsi yang ingin Anda tunjuk.
   d. (Opsional) Untuk mengonfigurasikan routing pada alias, perluas Alias tertimbang. Untuk informasi selengkapnya, lihat Menerapkan penerapan kenari Lambda menggunakan alias tertimbang.
   e. Pilih Simpan.

AWS CLI

Untuk membuat alias menggunakan AWS Command Line Interface (AWS CLI), gunakan perintah create-alias.

```
aws lambda create-alias \
  --function-name my-function \
  --name alias-name \
  --function-version version-number \
  --description " "
```

Untuk mengubah alias untuk mengarahkan versi baru fungsi, gunakan perintah update-alias.

```
aws lambda update-alias \
  --function-name my-function \
  --name alias-name \
  --function-version version-number
```

Untuk menghapus alias, gunakan perintah delete-alias.

```
aws lambda delete-alias \
  --function-name my-function \
  --name alias-name
```

AWS CLI Perintah pada langkah sebelumnya sesuai dengan operasi API Lambda berikut:

- CreateAlias
- UpdateAlias
- DeleteAlias

Menggunakan alias Lambda dalam sumber acara dan kebijakan izin

Setiap alias memiliki ARN yang unik. Alias hanya dapat menunjuk ke versi fungsi, bukan ke alias lain. Anda dapat memperbarui alias untuk menunjuk ke versi baru fungsi ini.

Sumber event seperti Amazon Simple Storage Service (Amazon S3) menginvokasi fungsi Lambda Anda. Sumber-sumber event ini mempertahankan pemetaan yang mengidentifikasi fungsi untuk diinvokasi ketika event terjadi. Jika Anda menentukan alias fungsi Lambda dalam konfigurasi pemetaan, Anda tidak perlu memperbarui pemetaan saat versi fungsi berubah. Untuk informasi selengkapnya, lihat Bagaimana Lambda memproses catatan dari sumber acara berbasis streaming dan antrian.

Dalam kebijakan sumber daya, Anda dapat memberikan izin kepada sumber event untuk menggunakan fungsi Lambda Anda. Jika Anda menentukan ARN alias dalam kebijakan, Anda tidak perlu memperbarui kebijakan ketika versi fungsi berubah.

Kebijakan sumber daya

Anda dapat menggunakan kebijakan berbasis sumber daya untuk memberikan layanan, sumber daya, atau akses akun ke fungsi Anda. Cakupan izin tersebut tergantung pada apakah Anda menerapkannya ke alias, versi, atau keseluruhan fungsi. Misalnya, jika Anda menggunakan nama alias (seperti`helloworld:PROD`), Anda dapat memanggil `helloworld` fungsi menggunakan alias ARN (`helloworld:PROD`).

Jika Anda mencoba memanggil fungsi tersebut tanpa alias atau versi tertentu, Anda mendapatkan kesalahan izin. Kesalahan izin ini masih terjadi meskipun Anda berupaya untuk memanggil secara langsung versi fungsi yang terkait dengan alias.

Misalnya, AWS CLI perintah berikut memberikan izin kepada Amazon S3 untuk memanggil alias PROD dari `helloworld` fungsi saat Amazon S3 bertindak atas nama. `amzn-s3-demo-bucket`

```
aws lambda add-permission \
  --function-name helloworld \
  --qualifier PROD \
  --statement-id 1 \
  --principal s3.amazonaws.com \
  --action lambda:InvokeFunction \
  --source-arn arn:aws:s3:::amzn-s3-demo-bucket \
  --source-account 123456789012
```

Untuk informasi selengkapnya tentang menggunakan nama sumber daya dalam kebijakan, lihat Fine-tuning Bagian Sumber Daya dan Ketentuan dari kebijakan.

Menerapkan penerapan kenari Lambda menggunakan alias tertimbang

Anda dapat menggunakan alias tertimbang untuk membagi lalu lintas antara dua versi berbeda dari fungsi yang sama. Dengan pendekatan ini, Anda dapat menguji versi baru dari fungsi Anda dengan persentase lalu lintas yang kecil dan dengan cepat memutar kembali jika perlu. Ini dikenal sebagai penyebaran kenari. Penerapan Canary berbeda dari blue/green penerapan dengan mengekspos versi baru hanya pada sebagian permintaan daripada mengalihkan semua lalu lintas sekaligus.

Anda dapat mengarahkan alias ke maksimal dua versi fungsi Lambda. Versi harus memenuhi kriteria berikut:

- Kedua versi harus memiliki peran eksekusi sama.
- Kedua versi harus memiliki konfigurasi antrean surat mati sama, atau tidak ada konfigurasi antrean surat mati.
- Kedua versi tersebut harus dipublikasikan. Alias tidak dapat menunjuk ke `$LATEST`.

> **Note**
>
> Lambda menggunakan model probabilistik sederhana untuk mendistribusikan lalu lintas di antara dua versi fungsi. Pada tingkat lalu lintas rendah, Anda mungkin melihat varians tinggi di antara persentase lalu lintas yang dikonfigurasi dan aktual di setiap versi. Jika fungsi Anda menggunakan konkurensi terprovisi, Anda dapat menghindari invokasi limpahan dengan mengonfigurasi jumlah yang lebih tinggi dari instans konkurensi terprovisi selama perutean alias aktif.

Buat alias tertimbang

Console

Untuk mengkonfigurasi routing pada alias menggunakan konsol

> **Note**
>
> Verifikasi bahwa fungsi memiliki setidaknya dua versi yang dipublikasikan. Untuk membuat versi tambahan, ikuti petunjuk dalam Membuat versi fungsi.

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih fungsi.
3. Pilih Alias, lalu pilih Buat alias.
4. Di halaman Buat alias, lakukan hal berikut:
   a. Masukkan Nama untuk alias.
   b. (Opsional) Masukkan Deskripsi untuk alias.
   c. Untuk Versi, pilih versi fungsi pertama yang ingin Anda tunjuk.
   d. Perluas Alias tertimbang.
   e. Untuk Versi tambahan, pilih versi fungsi kedua yang ingin Anda tunjuk.
   f. Untuk Bobot (%), masukkan nilai bobot untuk fungsi. Bobot adalah persentase lalu lintas yang ditetapkan untuk versi tersebut ketika alias dipanggil. Versi pertama menerima bobot residual. Misalnya, jika Anda menentukan 10 persen ke Versi tambahan, versi pertama ditetapkan ke 90 persen secara otomatis.
   g. Pilih Simpan.

AWS CLI

Gunakan AWS CLI perintah create-alias dan update-alias untuk mengonfigurasi bobot lalu lintas antara dua versi fungsi. Saat membuat atau memperbarui alias, Anda menentukan berat lalu lintas dalam parameter `routing-config`.

Contoh berikut membuat alias fungsi Lambda bernama routing-alias yang menunjuk ke versi 1 dari fungsi. Versi 2 fungsi menerima 3 persen lalu lintas. 97 persen lalu lintas yang tersisa diarahkan ke versi 1.

```
aws lambda create-alias \
  --name routing-alias \
  --function-name my-function \
  --function-version 1  \
  --routing-config AdditionalVersionWeights={"2"=0.03}
```

Gunakan perintah `update-alias` untuk meningkatkan persentase lalu lintas masuk ke versi 2. Dalam contoh berikut, Anda meningkatkan lalu lintas menjadi 5 persen.

```
aws lambda update-alias \
  --name routing-alias \
  --function-name my-function \
  --routing-config AdditionalVersionWeights={"2"=0.05}
```

Untuk mengirimkan semua lalu lintas ke versi 2, gunakan `update-alias` untuk mengubah properti `function-version` untuk menunjuk alias ke versi 2. Perintah juga mengatur ulang konfigurasi perutean.

```
aws lambda update-alias \
  --name routing-alias \
  --function-name my-function  \
  --function-version 2 \
  --routing-config AdditionalVersionWeights={}
```

AWS CLI Perintah pada langkah sebelumnya sesuai dengan operasi API Lambda berikut:

- CreateAlias
- UpdateAlias

Menentukan versi mana yang dipanggil

Saat Anda mengonfigurasi bobot lalu lintas di antara dua versi fungsi, ada dua cara untuk menentukan versi fungsi Lambda yang sudah dipanggil:

- CloudWatch Log — Lambda secara otomatis memancarkan entri `START` log yang berisi ID versi yang dipanggil untuk setiap pemanggilan fungsi. Contoh:

  ```
  START RequestId: 1dh194d3759ed-4v8b-a7b4-1e541f60235f Version: 2
  ```

Untuk invokasi alias, Lambda menggunakan dimensi `ExecutedVersion` untuk memfilter data metrik dengan versi yang dipanggil. Untuk informasi selengkapnya, lihat Melihat metrik untuk Fungsi Acara Lambda.

- Muatan respons (invokasi tersinkron) – Respons untuk invokasi fungsi tersinkron yang mencakup header `x-amz-executed-version` untuk menunjukkan versi fungsi mana yang telah dipanggil.

Buat penerapan bergulir dengan alias tertimbang

Use AWS CodeDeploy and AWS Serverless Application Model (AWS SAM) untuk membuat penerapan bergulir yang secara otomatis mendeteksi perubahan pada kode fungsi Anda, menerapkan versi baru fungsi Anda, dan secara bertahap meningkatkan jumlah lalu lintas yang mengalir ke versi baru. Jumlah lalu lintas dan tingkat peningkatan adalah parameter yang dapat Anda konfigurasi.

Dalam penerapan bergulir, AWS SAM lakukan tugas-tugas ini:

- Mengkonfigurasi fungsi Lambda Anda dan membuat alias. Konfigurasi routing alias tertimbang adalah kemampuan dasar yang mengimplementasikan penerapan bergulir.
- Membuat grup CodeDeploy aplikasi dan penyebaran. Grup penyebaran mengelola penerapan bergulir dan rollback, jika diperlukan.
- Mendeteksi saat Anda membuat versi baru fungsi Lambda Anda.
- Pemicu CodeDeploy untuk memulai penyebaran versi baru.

Contoh AWS SAM templat

Contoh berikut menunjukkan templat AWS SAM untuk deployment bergulir sederhana.

```
AWSTemplateFormatVersion : '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: A sample SAM template for deploying Lambda functions

Resources:
# Details about the myDateTimeFunction Lambda function
  myDateTimeFunction:
    Type: AWS::Serverless::Function
    Properties:
      Handler: myDateTimeFunction.handler
      Runtime: nodejs24.x
```

```
# Creates an alias named "live" for the function, and automatically publishes when you
 update the function.
      AutoPublishAlias: live
      DeploymentPreference:
# Specifies the deployment configuration
          Type: Linear10PercentEvery2Minutes
```

Templat ini menentukan fungsi Lambda bernama `myDateTimeFunction` dengan properti berikut.

AutoPublishAlias

Properti `AutoPublishAlias` membuat alias bernama `live`. Selain itu, kerangka kerja AWS SAM secara otomatis mendeteksi saat Anda menyimpan kode baru untuk fungsi tersebut. Kerangka kerja kemudian menerbitkan versi fungsi baru dan memperbarui alias `live` untuk mengarah ke versi baru.

DeploymentPreference

`DeploymentPreference`Properti menentukan tingkat di mana CodeDeploy aplikasi menggeser lalu lintas dari versi asli fungsi Lambda ke versi baru. Nilai tersebut `Linear10PercentEvery2Minutes` menggeser tambahan 10 persen lalu lintas ke versi baru setiap 2 menit.

Untuk daftar konfigurasi deployment yang ditetapkan sebelumnya, lihat Konfigurasi deployment.

Untuk informasi selengkapnya tentang cara membuat penerapan bergulir dengan CodeDeploy dan AWS SAM, lihat berikut ini:

- Tutorial: Menyebarkan fungsi CodeDeploy Lambda yang diperbarui dengan dan AWS Serverless Application Model
- Menerapkan aplikasi tanpa server secara bertahap dengan AWS SAM

Kelola versi Fungsi Acara Lambda

Anda dapat menggunakan versi untuk mengelola deployment fungsi Anda. Misalnya, Anda dapat menerbitkan versi baru dari fungsi untuk uji beta tanpa memengaruhi pengguna dari versi produksi stabil. Lambda membuat versi baru fungsi Anda setiap kali Anda menerbitkan fungsi. Versi baru adalah salinan dari versi fungsi yang belum diterbitkan. Versi yang tidak dipublikasikan diberi nama`$LATEST`.

Yang penting, setiap kali Anda menerapkan kode fungsi Anda, Anda menimpa kode saat ini. `$LATEST` Untuk menyimpan iterasi saat ini`$LATEST`, buat versi fungsi baru. Jika `$LATEST` identik dengan versi yang diterbitkan sebelumnya, Anda tidak akan dapat membuat versi baru sampai Anda menerapkan perubahan. `$LATEST` Perubahan ini dapat mencakup memperbarui kode, atau memodifikasi pengaturan konfigurasi fungsi.

Setelah Anda mempublikasikan versi fungsi, kode, runtime, arsitektur, memori, lapisan, dan sebagian besar pengaturan konfigurasi lainnya tidak dapat diubah. Ini berarti Anda tidak dapat mengubah pengaturan ini tanpa menerbitkan versi baru dari`$LATEST`. Anda dapat mengonfigurasi item berikut untuk versi fungsi yang diterbitkan:

- Pemicu
- Destinasi
- Konkurensi yang disediakan
- Pemanggilan asinkron
- Koneksi database dan proxy

> **Note**
>
> Saat menggunakan kontrol manajemen runtime dengan mode Otomatis, versi runtime yang digunakan oleh versi fungsi diperbarui secara otomatis. Saat menggunakan pembaruan Fungsi atau mode Manual, versi runtime tidak diperbarui. Untuk informasi selengkapnya, lihat the section called “Pembaruan versi runtime”.

Bagian-bagian

- Membuat versi fungsi
- Menggunakan versi

- Memberi izin

Membuat versi fungsi

Anda dapat mengubah kode dan pengaturan fungsi hanya pada versi fungsi yang belum diterbitkan. Saat Anda memublikasikan versi, Lambda mengunci kode dan sebagian besar pengaturan untuk mempertahankan pengalaman yang konsisten bagi pengguna versi tersebut.

Anda dapat membuat versi fungsi menggunakan konsol Lambda.

Untuk membuat versi baru fungsi

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih fungsi dan kemudian pilih tab Versi.
3. Pada halaman konfigurasi fungsi, pilih Terbitkan versi baru.
4. (Opsional) Masukkan deskripsi versi.
5. Pilih Terbitkan.

Atau, Anda dapat mempublikasikan versi fungsi menggunakan operasi PublishVersionAPI.

AWS CLI Perintah berikut menerbitkan versi baru dari suatu fungsi. Respons tersebut mengembalikan informasi konfigurasi tentang versi baru, termasuk nomor versi dan fungsi ARN dengan akhiran versi.

```
aws lambda publish-version --function-name my-function
```

Anda akan melihat output berikut:

```
{
  "FunctionName": "my-function",
  "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-function:1",
  "Version": "1",
  "Role": "arn:aws:iam::123456789012:role/lambda-role",
  "Handler": "function.handler",
  "Runtime": "nodejs24.x",
  ...
}
```

> **Note**
>
> Lambda menetapkan nomor urut yang meningkat secara monoton untuk pembuatan versi. Lambda tidak pernah menggunakan kembali nomor versi, bahkan setelah Anda menghapus dan membuat ulang fungsi.

Menggunakan versi

Anda dapat merujuk fungsi Lambda Anda menggunakan ARN yang memenuhi syarat atau ARN yang tidak memenuhi syarat.

- ARN yang Memenuhi Syarat – Fungsi ARN dengan akhiran versi. Contoh berikut mengacu pada versi 42 dari fungsi `helloworld`.

```
arn:aws:lambda:aws-region:acct-id:function:helloworld:42
```

- ARN yang Tidak Memenuhi Syarat – Fungsi ARN tanpa akhiran versi.

```
arn:aws:lambda:aws-region:acct-id:function:helloworld
```

Anda dapat menggunakan ARN yang memenuhi syarat atau tidak memenuhi syarat dalam semua operasi API terkait. Namun, Anda tidak dapat menggunakan ARN yang tidak memenuhi syarat untuk membuat alias.

Jika Anda memutuskan untuk tidak menerbitkan versi fungsi, Anda dapat mengaktifkan fungsi tersebut menggunakan ARN yang memenuhi syarat atau tidak memenuhi syarat dalam pemetaan sumber peristiwa. Saat Anda menjalankan fungsi menggunakan ARN yang tidak memenuhi syarat, Lambda secara implisit memanggil. `$LATEST`

ARN yang memenuhi syarat untuk setiap versi fungsi Lambda adalah unik. Setelah Anda menerbitkan versi, Anda tidak dapat mengubah ARN atau kode fungsi.

Lambda menerbitkan versi fungsi baru hanya jika kode belum pernah dipublikasikan, atau jika kode telah berubah dari versi terakhir yang diterbitkan. Jika tidak ada perubahan, versi fungsi tetap pada versi publikasi terakhir.

Saat Anda memublikasikan versi, Lambda membuat snapshot yang tidak dapat diubah dari kode dan konfigurasi fungsi Anda. Tidak semua perubahan konfigurasi memicu publikasi versi baru. Perubahan konfigurasi berikut memenuhi syarat fungsi untuk publikasi versi:

- Kode fungsi
- Variabel lingkungan
- Waktu Aktif
- Handler
- Lapisan
- Ukuran memori
- Batas waktu
- Konfigurasi VPC
- Konfigurasi Dead Letter Queue (DLQ)
- IAM Role
- Deskripsi
- Arsitektur (x86_64 atau arm64)
- Ukuran penyimpanan sementara
- Jenis Package
- Mode penyimpanan kode (beralih antara Lambda-managed dan penyimpanan S3 yang dikelola sendiri)
- Konfigurasi pencatatan log
- Konfigurasi sistem file
- SnapStart
- Konfigurasi penelusuran

Pengaturan operasional seperti konkurensi cadangan tidak memicu publikasi versi baru saat diubah.

Memberi izin

Anda dapat menggunakan kebijakan berbasis sumber daya atau kebijakan berbasis identitas untuk memberikan akses ke fungsi Anda. Ruang lingkup izin tergantung pada apakah Anda menerapkan kebijakan ke fungsi atau ke salah satu versi fungsi. Untuk informasi selengkapnya tentang nama sumber daya fungsi dalam kebijakan, lihat Fine-tuning Bagian Sumber Daya dan Ketentuan dari kebijakan.

Anda dapat menyederhanakan pengelolaan sumber peristiwa dan AWS Identity and Access Management (IAM) kebijakan dengan menggunakan alias fungsi. Lihat informasi yang lebih lengkap di Buat alias untuk Fungsi Acara Lambda.

Menggunakan tag pada Fungsi Acara Lambda

Anda dapat menandai fungsi untuk mengatur dan mengelola sumber daya Anda. Tag adalah pasangan nilai kunci bentuk bebas yang terkait dengan sumber daya Anda yang didukung secara keseluruhan. Layanan AWS Untuk informasi selengkapnya tentang kasus penggunaan tag, lihat Strategi penandaan umum di AWS Sumber Daya Penandaan dan Panduan Editor Tag.

Tanda berlaku pada tingkat fungsi, bukan pada versi atau alias. Tag bukan bagian dari konfigurasi khusus versi yang AWS Lambda membuat snapshot saat Anda memublikasikan versi. Anda dapat menggunakan API Lambda untuk melihat dan memperbarui tag. Anda juga dapat melihat dan memperbarui tag sambil mengelola fungsi tertentu di konsol Lambda.

Anda juga dapat menggunakan tag untuk mengontrol akses ke fungsi Anda dengan kontrol akses berbasis atribut (ABAC). Untuk informasi selengkapnya, lihat the section called “Attribute-based kontrol akses”.

Bagian-bagian

- Izin diperlukan untuk bekerja dengan tag
- Menggunakan tag dengan konsol Lambda
- Menggunakan tag dengan AWS CLI

Izin diperlukan untuk bekerja dengan tag

Untuk mengizinkan identitas AWS Identity and Access Management (IAM) (pengguna, grup, atau peran) membaca atau menyetel tag pada sumber daya, berikan izin yang sesuai:

- lambda: ListTags —Ketika sumber daya memiliki tag, berikan izin ini kepada siapa saja yang perlu memanggilnya`ListTags`. Untuk Fungsi Acara yang ditandai, izin ini juga diperlukan untuk`GetFunction`.
- lambda: TagResource —Berikan izin ini kepada siapa saja yang perlu memanggil `TagResource` atau melakukan tag saat membuat.

Secara opsional, pertimbangkan untuk memberikan UntagResource izin lambda: juga untuk mengizinkan `UntagResource` panggilan ke sumber daya.

Untuk informasi selengkapnya, lihat Identity-based Kebijakan IAM untuk Lambda.

Menggunakan tag dengan konsol Lambda

Anda dapat menggunakan konsol Lambda untuk membuat fungsi yang memiliki tag, menambahkan tag ke fungsi yang ada, dan memfilter fungsi berdasarkan tag yang Anda tambahkan.

Untuk menambahkan tag saat Anda membuat fungsi

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih Buat Fungsi Acara.
3. Pilih Penulis dari awal atau gambar Kontainer.
4. Di bawah Informasi dasar, atur fungsi Anda. Untuk informasi selengkapnya tentang mengonfigurasi fungsi, lihatthe section called “Mengkonfigurasi Fungsi Acara”.
5. Perluas Pengaturan lanjutan, lalu pilih Aktifkan tag.
6. Pilih Tambahkan tag baru, lalu masukkan Kunci dan Nilai opsional. Untuk menambahkan lebih banyak tag, ulangi langkah ini.
7. Pilih Buat Fungsi Acara.

Untuk menambahkan tag ke fungsi yang ada

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih nama sebuah fungsi.
3. Pilih Konfigurasi, lalu pilih Tag.
4. Di bagian Tanda, pilih Kelola tanda.
5. Pilih Tambahkan tag baru, lalu masukkan Kunci dan Nilai opsional. Untuk menambahkan lebih banyak tag, ulangi langkah ini.
6. Pilih Simpan.

Untuk memfilter fungsi dengan tanda

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih kotak pencarian untuk melihat daftar properti fungsi dan tombol tag.
3. Pilih kunci tag untuk melihat daftar nilai yang sedang digunakan di AWS Wilayah saat ini.
4. Pilih Gunakan: “tag-name” untuk melihat semua fungsi yang ditandai dengan kunci ini, atau pilih Operator untuk memfilter lebih lanjut berdasarkan nilai.

5. Pilih nilai tag Anda untuk difilter dengan kombinasi kunci tag dan nilai.

Bilah pencarian juga mendukung pencarian kunci tanda. Masukkan `tag` untuk melihat hanya daftar kunci tag, atau masukkan nama kunci untuk menemukannya dalam daftar.

Menggunakan tag dengan AWS CLI

Anda dapat menambahkan dan menghapus tag pada sumber daya Lambda yang ada, termasuk fungsi, dengan API Lambda. Anda juga dapat menambahkan tag saat membuat fungsi, untuk menjaga sumber daya diberi tag melalui seluruh siklus hidupnya.

Memperbarui tag dengan API tag Lambda

Anda dapat menambahkan dan menghapus tag untuk sumber daya Lambda yang didukung melalui operasi TagResourcedan UntagResourceAPI.

Anda dapat memanggil operasi ini menggunakan AWS CLI. Untuk menambahkan tag ke sumber daya yang ada, gunakan `tag-resource` perintah. Contoh ini menambahkan dua tag, satu dengan kunci *`Department`* dan satu dengan kunci*`CostCenter`*.

```
aws lambda tag-resource \
--resource arn:aws:lambda:us-east-2:123456789012:resource-type:my-resource \
--tags Department=Marketing,CostCenter=1234ABCD
```

Untuk menghapus tanda, gunakan perintah `untag-resource`. Contoh ini menghapus tag dengan kunci*`Department`*.

```
aws lambda untag-resource --resource arn:aws:lambda:us-east-1:123456789012:resource-
type:resource-identifier \
--tag-keys Department
```

Menambahkan tag saat membuat fungsi

Untuk membuat fungsi Lambda baru dengan tag, gunakan operasi CreateFunctionAPI. Tentukan parameter `Tags`. Anda dapat memanggil operasi ini dengan perintah `create-function` CLI dan opsi --tags.

Sebelum menggunakan parameter tag dengan`CreateFunction`, pastikan bahwa peran Anda memiliki izin untuk menandai sumber daya bersama izin yang biasa diperlukan untuk operasi ini.

Untuk informasi selengkapnya tentang izin untuk menandai, lihat. the section called “Izin diperlukan untuk bekerja dengan tag” Contoh ini menambahkan dua tag, satu dengan kunci *Department* dan satu dengan kunci*CostCenter*.

```
aws lambda create-function --function-name my-function
--handler index.js --runtime nodejs24.x \
--role arn:aws:iam::123456789012:role/lambda-role \
--tags Department=Marketing,CostCenter=1234ABCD
```

Melihat tag pada suatu fungsi

Untuk melihat tag yang diterapkan ke sumber daya Lambda tertentu, gunakan operasi `ListTags` API. Untuk informasi selengkapnya, lihat ListTags.

Anda dapat memanggil operasi ini dengan `list-tags` AWS CLI perintah dengan memberikan ARN (Nama Sumber Daya Amazon).

```
aws lambda list-tags --resource arn:aws:lambda:us-east-1:123456789012:resource-
type:resource-identifier
```

Anda dapat melihat tag yang diterapkan ke sumber daya tertentu dengan operasi GetFunctionAPI. Fungsionalitas yang sebanding tidak tersedia untuk jenis sumber daya lainnya.

Anda dapat memanggil operasi ini dengan perintah `get-function` CLI:

```
aws lambda get-function --function-name my-function
```

Memfilter sumber daya berdasarkan tag

Anda dapat menggunakan operasi AWS Resource Groups Tagging API GetResourcesAPI untuk memfilter sumber daya Anda berdasarkan tag. `GetResources`Operasi menerima hingga 10 filter, dengan setiap filter berisi kunci tag dan hingga 10 nilai tag. Anda `GetResources` menyediakan `ResourceType` untuk memfilter berdasarkan jenis sumber daya tertentu.

Anda dapat memanggil operasi ini menggunakan `get-resources` AWS CLI perintah. Untuk contoh penggunaan`get-resources`, lihat mendapatkan sumber daya di Referensi Perintah AWS CLI.

Streaming respons untuk Fungsi Acara Lambda

Fungsi Lambda secara native dapat mengalirkan muatan respons kembali ke klien melalui URL fungsi Lambda atau dengan menggunakan InvokeWithResponseStreamAPI (melalui SDK atau panggilan API langsung). AWS Fungsi Lambda Anda juga dapat mengalirkan muatan respons melalui integrasi proxy Amazon API Gateway, yang menggunakan InvokeWithResponseStreamAPI untuk menjalankan fungsi Anda. Streaming respons dapat menguntungkan aplikasi sensitif latensi dengan meningkatkan kinerja time to first byte (TTFB). Ini karena Anda dapat mengirim sebagian tanggapan kembali ke klien saat tersedia. Selain itu, fungsi streaming respons dapat mengembalikan muatan hingga 200 MB, dibandingkan dengan maksimum 6 MB untuk respons buffer.

Streaming respons juga berarti bahwa fungsi Anda tidak perlu sesuai dengan seluruh respons dalam memori. Untuk respons yang sangat besar, ini dapat mengurangi jumlah memori yang perlu Anda konfigurasikan untuk fungsi Anda. Untuk membantu memilih antara URL fungsi dan API Gateway untuk kasus penggunaan Anda, lihatthe section called “URL Fungsi vs Amazon API Gateway”.

> **Note**
>
> Streaming respons Lambda belum tersedia di semua. Wilayah AWS Untuk ketersediaan fitur menurut Wilayah, lihat AWS Kemampuan Builder Center menurut Wilayah.

Kecepatan Lambda mengalirkan respons Anda tergantung pada ukuran respons. Rasio streaming untuk 6 MB pertama dari respons fungsi Anda tidak dibatasi. Untuk respons yang lebih besar dari 6 MB, sisa respons tunduk pada batas bandwidth. Untuk informasi lebih lanjut tentang bandwidth streaming, lihatBatas bandwidth untuk streaming respons.

Respons streaming menimbulkan biaya dan tanggapan yang dialirkan tidak terganggu atau dihentikan ketika koneksi klien yang memanggil terputus. Pelanggan ditagih untuk durasi fungsi penuh, jadi pelanggan harus berhati-hati saat mengonfigurasi batas waktu fungsi yang lama.

Lambda mendukung streaming respons pada runtime Node.js terkelola. Untuk bahasa lain, termasuk Python, Anda dapat menggunakan runtime kustom dengan integrasi API Runtime kustom untuk melakukan streaming respons atau menggunakan Adaptor Web Lambda.

> **Note**
>
> Saat menguji fungsi Anda melalui konsol Lambda, Anda akan selalu melihat respons sebagai buffer.

Untuk mempelajari cara menulis fungsi streaming, lihatthe section called “Fungsi menulis”. Untuk menjalankan fungsi streaming melalui URL fungsi, lihat. the section called “Memanggil fungsi” Untuk panduan langkah demi langkah, lihat. the section called “Tutorial: Membuat fungsi streaming respons dengan URL fungsi”

Topik

- Batas bandwidth untuk streaming respons
- Kompatibilitas VPC dengan streaming respons
- Menulis fungsi Lambda yang mendukung streaming respons
- Menginvokasi Fungsi yang Diaktifkan Streaming Respons Menggunakan URL Fungsi Lambda
- Tutorial: Membuat Fungsi Lambda Streaming Respons dengan URL Fungsi

Batas bandwidth untuk streaming respons

6 MB pertama dari payload respons fungsi Anda memiliki bandwidth yang tidak dibatasi. Setelah ledakan awal ini, Lambda mengalirkan respons Anda dengan kecepatan maksimum 2 MBps. Jika respons fungsi Anda tidak pernah melebihi 6 MB, maka batas bandwidth ini tidak pernah berlaku.

> **Note**
>
> Batas bandwidth hanya berlaku untuk payload respons fungsi Anda, dan bukan untuk akses jaringan oleh fungsi Anda.

Tingkat bandwidth yang tidak dibatasi bervariasi tergantung pada sejumlah faktor, termasuk kecepatan pemrosesan fungsi Anda. Anda biasanya dapat mengharapkan tingkat yang lebih tinggi dari 2 MBps untuk 6 MB pertama dari respons fungsi Anda. Jika fungsi Anda mengalirkan respons ke tujuan di luar AWS, kecepatan streaming juga tergantung pada kecepatan koneksi internet eksternal.

Kompatibilitas VPC dengan streaming respons

Saat menggunakan fungsi Lambda di lingkungan VPC, ada pertimbangan penting untuk streaming respons:

- URL fungsi Lambda tidak mendukung streaming respons dalam lingkungan VPC.
- Anda dapat menggunakan streaming respons dalam VPC dengan menjalankan fungsi Lambda Anda melalui SDK menggunakan API. AWS `InvokeWithResponseStream` Ini memerlukan pengaturan titik akhir VPC yang sesuai untuk Lambda.
- Untuk lingkungan VPC, Anda harus membuat titik akhir VPC antarmuka untuk Lambda untuk memungkinkan komunikasi antara sumber daya Anda di VPC dan layanan Lambda.

Arsitektur khas untuk streaming respons dalam VPC mungkin termasuk:

```
Client in VPC -> Interface VPC endpoint for Lambda -> Lambda function -> Response
 streaming back through the same path
```

Menulis fungsi Lambda yang mendukung streaming respons

Menulis handler untuk fungsi streaming respons berbeda dari pola handler biasa. Saat menulis fungsi streaming, pastikan untuk melakukan hal berikut:

- Bungkus fungsi Anda dengan `awslambda.streamifyResponse()` dekorator. Objek `awslambda` global disediakan oleh lingkungan runtime Node.js Lambda.
- Akhiri aliran dengan anggun untuk memastikan bahwa semua pemrosesan data selesai.

Mengkonfigurasi fungsi handler untuk mengalirkan respons

Untuk menunjukkan ke runtime bahwa Lambda harus mengalirkan respons fungsi Anda, Anda harus membungkus fungsi Anda dengan `streamifyResponse()` dekorator. Ini memberi tahu runtime untuk menggunakan jalur logika yang tepat untuk respons streaming dan memungkinkan fungsi untuk mengalirkan respons.

`streamifyResponse()`Dekorator menerima fungsi yang menerima parameter berikut:

- `event`— Memberikan informasi tentang peristiwa pemanggilan URL fungsi, seperti metode HTTP, parameter kueri, dan badan permintaan.
- `responseStream`— Menyediakan aliran yang dapat ditulis.

- `context`Menyediakan metode dan properti dengan informasi tentang pemanggilan, fungsi, dan lingkungan eksekusi.

`responseStream`Objeknya adalah Node.js `writableStream`. Seperti halnya aliran seperti itu, Anda harus menggunakan `pipeline()` metode ini.

> **Note**
>
> Objek `awslambda` global secara otomatis disediakan oleh runtime Node.js Lambda dan tidak diperlukan impor.

Example handler yang mendukung streaming respons

```
import { pipeline } from 'node:stream/promises';
import { Readable } from 'node:stream';

export const echo = awslambda.streamifyResponse(async (event, responseStream, _context)
 => {
  // As an example, convert event to a readable stream.
  const requestStream = Readable.from(Buffer.from(JSON.stringify(event)));

  await pipeline(requestStream, responseStream);
});
```

Meskipun `responseStream` menawarkan `write()` metode untuk menulis ke aliran, kami sarankan Anda menggunakannya `pipeline()`sedapat mungkin. Menggunakan `pipeline()` memastikan bahwa aliran yang dapat ditulis tidak kewalahan oleh aliran yang dapat dibaca lebih cepat.

Mengakhiri aliran

Pastikan Anda mengakhiri aliran dengan benar sebelum handler kembali. `pipeline()`Metode ini menangani ini secara otomatis.

Untuk kasus penggunaan lainnya, panggil `responseStream.end()` metode untuk mengakhiri aliran dengan benar. Metode ini menandakan bahwa tidak ada lagi data yang harus ditulis ke aliran. Metode ini tidak diperlukan jika Anda menulis ke aliran dengan `pipeline()` ataupipe().

Dimulai dengan Node.js 24, Lambda tidak lagi menunggu janji yang belum terselesaikan selesai setelah handler Anda kembali atau aliran respons berakhir. Jika fungsi Anda bergantung pada

operasi asinkron tambahan, seperti pengatur waktu atau pengambilan, Anda harus `await` menggunakannya di handler Anda.

Example Contoh mengakhiri aliran dengan pipeline ()

```
import { pipeline } from 'node:stream/promises';

export const handler = awslambda.streamifyResponse(async (event, responseStream,
 _context) => {
  await pipeline(requestStream, responseStream);
});
```

Example Contoh mengakhiri aliran tanpa pipeline ()

```
export const handler = awslambda.streamifyResponse(async (event, responseStream,
 _context) => {
  responseStream.write("Hello ");
  responseStream.write("world ");
  responseStream.write("from ");
  responseStream.write("Lambda!");
  responseStream.end();
});
```

Menginvokasi Fungsi yang Diaktifkan Streaming Respons Menggunakan URL Fungsi Lambda

> **Note**
>
> Fungsi Lambda Anda sekarang dapat mengalirkan muatan respons melalui integrasi proxy Amazon API Gateway.

Anda dapat memanggil fungsi yang diaktifkan streaming respons dengan mengubah mode pemanggilan URL fungsi Anda. Mode pemanggilan menentukan operasi API yang digunakan Lambda untuk menjalankan fungsi Anda. Mode pemanggilan yang tersedia adalah:

- `BUFFERED`— Ini adalah opsi default. Lambda memanggil fungsi Anda menggunakan operasi API. `Invoke` Hasil pemanggilan tersedia saat muatan selesai. Ukuran muatan maksimum adalah 6 MB.

- `RESPONSE_STREAM`— Memungkinkan fungsi Anda untuk mengalirkan hasil payload saat tersedia. Lambda memanggil fungsi Anda menggunakan operasi API. `InvokeWithResponseStream` Ukuran payload respons maksimum adalah 200 MB.

Anda masih dapat menjalankan fungsi Anda tanpa streaming respons dengan langsung memanggil operasi `Invoke` API. Namun, Lambda mengalirkan semua muatan respons untuk pemanggilan yang datang melalui URL fungsi hingga Anda mengubah mode pemanggilan menjadi. `BUFFERED`

Console

Untuk mengatur mode pemanggilan URL fungsi (konsol)

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih nama fungsi yang ingin Anda atur untuk mode pemanggilan.
3. Pilih tab Konfigurasi, lalu pilih URL Fungsi.
4. Pilih Edit, lalu pilih Pengaturan tambahan.
5. Di bawah mode Panggilan, pilih mode pemanggilan yang Anda inginkan.
6. Pilih Simpan.

AWS CLI

Untuk mengatur mode pemanggilan URL ()AWS CLI fungsi

```
aws lambda update-function-url-config \
  --function-name my-function \
  --invoke-mode RESPONSE_STREAM
```

CloudFormation

Untuk mengatur mode pemanggilan URL ()CloudFormation fungsi

```
MyFunctionUrl:
  Type: AWS::Lambda::Url
  Properties:
    AuthType: AWS_IAM
    InvokeMode: RESPONSE_STREAM
```

Untuk informasi selengkapnya tentang mengonfigurasi URL fungsi, lihat URL fungsi Lambda.

Tutorial: Membuat Fungsi Lambda Streaming Respons dengan URL Fungsi

Dalam tutorial ini, Anda membuat fungsi Lambda yang didefinisikan sebagai arsip file.zip dengan titik akhir URL fungsi yang mengembalikan aliran respons. Untuk informasi selengkapnya tentang mengonfigurasi URL fungsi, lihat. URL fungsi

Prasyarat

Tutorial ini mengasumsikan bahwa Anda memiliki pengetahuan tentang operasi Lambda dan konsol Lambda dasar. Jika belum, ikuti petunjuk di Membuat fungsi Lambda dengan konsol untuk membuat fungsi Lambda pertama Anda.

Untuk menyelesaikan langkah-langkah berikut, Anda memerlukan AWS CLI versi 2. Perintah dan output yang diharapkan dicantumkan dalam blok terpisah:

```
aws --version
```

Anda akan melihat output berikut:

```
aws-cli/2.13.27 Python/3.11.6 Linux/4.14.328-248.540.amzn2.x86_64 exe/x86_64.amzn.2
```

Untuk perintah panjang, karakter escape (\) digunakan untuk memisahkan perintah menjadi beberapa baris.

Di Linux dan macOS, gunakan shell dan manajer paket pilihan Anda.

> **Note**
>
> Di Windows, beberapa perintah Bash CLI yang biasa Anda gunakan dengan Lambda (`zip`seperti) tidak didukung oleh terminal bawaan sistem operasi. Untuk mendapatkan Windows-integrated versi Ubuntu dan Bash, instal Subsistem Windows untuk Linux. Contoh perintah CLI dalam panduan ini menggunakan pemformatan Linux. Perintah yang menyertakan dokumen JSON sebaris harus diformat ulang jika Anda menggunakan CLI Windows.

Membuat peran eksekusi

Buat peran eksekusi yang memberikan izin kepada fungsi Lambda Anda untuk mengakses sumber daya AWS .

Untuk membuat peran eksekusi

1. Buka halaman Peran konsol AWS Identity and Access Management (IAM).
2. Pilih Buat peran.
3. Buat peran dengan properti berikut:
   - Jenis entitas tepercaya - AWS layanan
   - Kasus penggunaan - Lambda
   - Izin – AWSLambdaBasicExecutionRole
   - Nama peran – **`response-streaming-role`**

AWSLambdaBasicExecutionRoleKebijakan ini memiliki izin yang diperlukan fungsi untuk menulis log ke Amazon CloudWatch Logs. Setelah Anda membuat peran, catat Nama Sumber Daya Amazon (ARN). Anda akan membutuhkannya di langkah berikutnya.

Buat fungsi streaming respons (AWS CLI)

Buat respons streaming fungsi Lambda dengan titik akhir URL fungsi menggunakan (). AWS Command Line Interface AWS CLI

Untuk membuat fungsi yang dapat mengalirkan respons

1. Salin contoh kode berikut ke file bernama `index.js`. Fungsi ini mengalirkan tiga respons, dipisahkan oleh 1 detik.

```
exports.handler = awslambda.streamifyResponse(
 async (event, responseStream, _context) => {
  // Metadata is a JSON serializable JS object. Its shape is not defined here.
  const metadata = {
  statusCode: 200,
  headers: {
   "Content-Type": "application/json",
   "CustomHeader": "outerspace"
  }
  };

  // Assign to the responseStream parameter to prevent accidental reuse of the non-
wrapped stream.
  responseStream = awslambda.HttpResponseStream.from(responseStream, metadata);
```

```
  responseStream.write("Streaming with Helper \n");
  await new Promise(r => setTimeout(r, 1000));
  responseStream.write("Hello 0 \n");
  await new Promise(r => setTimeout(r, 1000));
  responseStream.write("Hello 1 \n");
  await new Promise(r => setTimeout(r, 1000));
  responseStream.write("Hello 2 \n");
  await new Promise(r => setTimeout(r, 1000));
  responseStream.end();
  await responseStream.finished();
 }
  );
```

2. Buat paket deployment.

```
zip function.zip index.js
```

3. Buat fungsi Lambda dengan perintah `create-function`. Ganti nilai `--role` dengan peran ARN dari langkah sebelumnya. Perintah ini menetapkan batas waktu fungsi menjadi 10 detik, yang memungkinkan fungsi untuk mengalirkan tiga respons.

```
aws lambda create-function \
  --function-name my-streaming-function \
  --runtime nodejs24.x \
  --zip-file fileb://function.zip \
  --handler index.handler \
  --timeout 10 \
  --role arn:aws:iam::123456789012:role/response-streaming-role
```

Untuk membuat URL fungsi

1. Tambahkan kebijakan berbasis sumber daya ke fungsi Anda yang memberikan dan izin. `lambda:InvokeFunctionUrl` `lambda:InvokeFunction` Setiap pernyataan harus ditambahkan dalam perintah terpisah. Ganti nilai `--principal` dengan Akun AWS ID Anda.

```
aws lambda add-permission \
  --function-name my-streaming-function \
  --action lambda:InvokeFunctionUrl \
  --statement-id UrlPolicyInvokeURL \
  --principal 123456789012 \
  --function-url-auth-type AWS_IAM
```

```
aws lambda add-permission \
    --function-name my-streaming-function \
    --action lambda:InvokeFunction \
    --statement-id UrlPolicyInvokeFunction \
    --principal 123456789012
```

2. Buat titik akhir URL untuk fungsi dengan `create-function-url-config` perintah.

```
aws lambda create-function-url-config \
  --function-name my-streaming-function \
  --auth-type AWS_IAM \
  --invoke-mode RESPONSE_STREAM
```

> **Note**
>
> Jika Anda mendapatkan kesalahan`--invoke-mode`, Anda mungkin perlu memutakhirkan ke versi yang lebih baru. AWS CLI

Uji titik akhir URL fungsi

Uji integrasi Anda dengan menjalankan fungsi Anda. Anda dapat membuka URL fungsi Anda di browser, atau Anda dapat menggunakan curl.

```
curl --request GET "https://abcdefghijklm7nop7qrs740abcd.lambda-url.us-east-1.on.aws/"
 --user "AKIAIOSFODNN7EXAMPLE" --aws-sigv4 "aws:amz:us-east-1:lambda" --no-buffer
```

URL fungsi kami menggunakan jenis `IAM_AUTH` otentikasi. Ini berarti Anda perlu menandatangani permintaan dengan kunci AWS akses dan kunci rahasia Anda. Pada perintah sebelumnya, ganti `AKIAIOSFODNN7EXAMPLE` dengan ID kunci AWS akses. Masukkan kunci AWS rahasia Anda saat diminta. Jika Anda tidak memiliki kunci AWS rahasia Anda, Anda dapat menggunakan AWS kredensyal sementara sebagai gantinya.

Anda akan melihat tanggapan seperti ini:

```
Streaming with Helper
Hello 0
Hello 1
Hello 2
```

Bersihkan sumber daya Anda

Sekarang Anda dapat menghapus sumber daya yang Anda buat untuk tutorial ini, kecuali Anda ingin mempertahankannya. Dengan menghapus AWS sumber daya yang tidak lagi Anda gunakan, Anda mencegah tagihan yang tidak perlu ke Anda Akun AWS.

Untuk menghapus peran eksekusi

1. Buka halaman Peran dari konsol IAM.
2. Pilih peran eksekusi yang Anda buat.
3. Pilih Hapus.
4. Masukkan nama peran di bidang input teks dan pilih Hapus.

Untuk menghapus Fungsi Acara Lambda

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih Fungsi Acara yang Anda buat.
3. Pilih Tindakan, Hapus.
4. Ketik **`confirm`** kolom input teks dan pilih Hapus.

Menggunakan titik akhir metadata Lambda

Dengan titik akhir metadata Lambda, fungsi Anda dapat menemukan Availability Zone (AZ) mana yang mereka jalankan. Anda dapat menggunakan ini untuk mengoptimalkan latensi dengan merutekan ke sumber daya AZ yang sama seperti titik akhir Amazon dan ElastiCache Amazon RDS, dan untuk menerapkan pola ketahanan. AZ-aware

Titik akhir mengembalikan metadata dalam format JSON sederhana melalui API HTTP localhost dalam lingkungan eksekusi dan dapat diakses oleh runtime dan ekstensi.

Bagian-bagian

- Memulai
- Memahami ID Zona Ketersediaan
- Referensi API

Memulai

Powertools untuk AWS Lambda menyediakan utilitas untuk mengakses titik akhir metadata Lambda di Python,, Java, dan .NET. TypeScript Utilitas menyimpan respons setelah panggilan pertama dan menangani pembatalan SnapStart cache secara otomatis.

Gunakan utilitas metadata Powertools untuk AWS Lambda atau hubungi titik akhir metadata secara langsung

Python

Instal paket Powertools:

```
pip install "aws-lambda-powertools"
```

Gunakan utilitas metadata di handler Anda:

Example Mengambil ID AZ dengan Powertools (Python)

```
from aws_lambda_powertools.utilities.lambda_metadata import get_lambda_metadata

def handler(event, context):
    metadata = get_lambda_metadata()
```

```
    az_id = metadata.availability_zone_id  # e.g., "use1-az1"

    return {"az_id": az_id}
```

TypeScript

Instal paket Powertools:

```
npm install @aws-lambda-powertools/commons
```

Gunakan utilitas metadata di handler Anda:

Example Mengambil ID AZ dengan Powertools () TypeScript

```
import { getMetadata } from '@aws-lambda-powertools/commons/utils/metadata';

const metadata = await getMetadata();

export const handler = async () => {
  const { AvailabilityZoneID: azId } = metadata;
  return azId;
};
```

Java

Tambahkan dependensi Powertools ke: `pom.xml`

```
<dependencies>
    <dependency>
        <groupId>software.amazon.lambda</groupId>
        <artifactId>powertools-lambda-metadata</artifactId>
        <version>2.10.0</version>
    </dependency>
</dependencies>
```

Gunakan klien metadata di handler Anda:

Example Mengambil ID AZ dengan Powertools (Java)

```
import software.amazon.lambda.powertools.metadata.LambdaMetadata;
```

```
import software.amazon.lambda.powertools.metadata.LambdaMetadataClient;

public class App implements RequestHandler<Object, String> {

    @Override
    public String handleRequest(Object input, Context context) {
        LambdaMetadata metadata = LambdaMetadataClient.get();
        String azId = metadata.getAvailabilityZoneId(); // e.g., "use1-az1"

        return "{\"azId\": \"" + azId + "\"}";
    }
}
```

.NET

Instal paket Powertools:

```
dotnet add package AWS.Lambda.Powertools.Metadata
```

Gunakan kelas metadata di handler Anda:

Example Mengambil ID AZ dengan Powertools (.NET)

```
using AWS.Lambda.Powertools.Metadata;

public class Function
{
    public string Handler(object input, ILambdaContext context)
    {
        var azId = LambdaMetadata.AvailabilityZoneId;
        return $"Running in AZ: {azId}";
    }
}
```

All Runtimes

Semua runtime Lambda mendukung titik akhir metadata, termasuk runtime kustom dan gambar kontainer. Gunakan contoh berikut untuk mengakses API metadata langsung dari fungsi Anda menggunakan variabel lingkungan yang secara otomatis ditetapkan Lambda di lingkungan eksekusi.

Example Mengakses titik akhir metadata secara langsung

```
# Variables are automatically set by Lambda
METADATA_ENDPOINT="http://${AWS_LAMBDA_METADATA_API}/2026-01-15/metadata/execution-
environment"

# Make the request
RESPONSE=$(curl -s -H "Authorization: Bearer ${AWS_LAMBDA_METADATA_TOKEN}"
 "$METADATA_ENDPOINT")

# Parse the AZ ID
AZ_ID=$(echo "$RESPONSE" | jq -r '.AvailabilityZoneID')

echo "Function is running in AZ ID: $AZ_ID"
```

Memahami ID Zona Ketersediaan

ID AZ (misalnya,`use1-az1`) selalu merujuk ke lokasi fisik yang sama di semua AWS akun, sementara nama AZ (misalnya,`us-east-1a`) mungkin dipetakan ke infrastruktur fisik yang berbeda di setiap AWS akun di wilayah tertentu. Untuk informasi selengkapnya, lihat ID AZ untuk konsistensi lintas akun.

Mengonversi ID AZ ke nama AZ:

Untuk mengonversi ID AZ ke nama AZ, gunakan Amazon EC2 API DescribeAvailabilityZones. Untuk menggunakan API ini, tambahkan `ec2:DescribeAvailabilityZones` izin ke peran eksekusi fungsi Anda.

Referensi API

Variabel lingkungan

Lambda secara otomatis menetapkan variabel lingkungan berikut di setiap lingkungan eksekusi:

- `AWS_LAMBDA_METADATA_API`— Alamat server metadata dalam format `{ipv4_address}:{port}` (misalnya,`169.254.100.1:9001`).
- `AWS_LAMBDA_METADATA_TOKEN`— Token otentikasi unik untuk lingkungan eksekusi saat ini. Lambda menghasilkan token ini secara otomatis saat inisialisasi. Sertakan dalam semua permintaan API metadata.

Titik akhir

`GET http://${AWS_LAMBDA_METADATA_API}/2026-01-15/metadata/execution-environment`

Permintaan

Header yang diperlukan:

- `Authorization`— Nilai token dari variabel `AWS_LAMBDA_METADATA_TOKEN` lingkungan dengan skema Bearer:`Bearer <token>`. Otentikasi berbasis token ini memberikan perlindungan mendalam terhadap kerentanan Server-Side Request Forgery (SSRF). Setiap lingkungan eksekusi menerima token unik yang dihasilkan secara acak saat inisialisasi.

Respons

Status: `200 OK`

Content-Type: `application/json`

Cache-Control: `private, max-age=43200, immutable`

Responsnya tidak dapat diubah dalam lingkungan eksekusi. Klien harus menyimpan respons dan menghormati `Cache-Control` TTL. Untuk SnapStart fungsi, TTL dikurangi selama inisialisasi sehingga klien menyegarkan metadata setelah pemulihan ketika lingkungan eksekusi mungkin berada di AZ yang berbeda. Jika Anda menggunakan Powertools, caching dan SnapStart invalidasi ditangani secara otomatis.

Tubuh:

```
{
  "AvailabilityZoneID": "use1-az1"
}
```

`AvailabilityZoneID`Bidang berisi pengenal unik untuk Availability Zone tempat lingkungan eksekusi berjalan.

> **Note**
>
> Bidang tambahan mungkin ditambahkan ke respons di pembaruan masa depan. Klien harus mengabaikan bidang yang tidak dikenal dan tidak gagal jika bidang baru muncul.

Respons kesalahan

- 401 Tidak Sah — `Authorization` Header tidak ada atau berisi token yang tidak valid. Verifikasi Anda lewat`Bearer ${AWS_LAMBDA_METADATA_TOKEN}`.
- 405 Metode Tidak Diizinkan - Metode permintaan tidak`GET`.
- 500 Kesalahan Server Internal - kesalahan Server-side pemrosesan.

Memahami metode pemanggilan Fungsi Peristiwa Lambda

Setelah Anda menerapkan Fungsi Peristiwa Lambda, Anda dapat memanggilnya dengan beberapa cara:

- Konsol Lambda — Gunakan konsol Lambda untuk membuat acara pengujian dengan cepat untuk menjalankan fungsi Anda.
- AWS SDK — Gunakan SDK untuk AWS menjalankan fungsi Anda secara terprogram.
- API Invoke — Gunakan Lambda Invoke API untuk langsung menjalankan fungsi Anda.
- The AWS Command Line Interface (AWS CLI) — Gunakan `aws lambda invoke` AWS CLI perintah untuk langsung memanggil fungsi Anda dari baris perintah.
- Titik akhir HTTP (S) URL fungsi — Gunakan URL fungsi untuk membuat titik akhir HTTP (S) khusus yang dapat Anda gunakan untuk menjalankan fungsi Anda.

Semua metode ini adalah cara langsung untuk memanggil Fungsi Acara Anda. Di Lambda, kasus penggunaan umum adalah memanggil fungsi Anda berdasarkan peristiwa yang terjadi di tempat lain dalam aplikasi Anda. Beberapa layanan dapat memanggil Fungsi Acara Lambda dengan setiap acara baru.

Ini disebut pemicu. Untuk layanan berbasis streaming dan antrian, Lambda memanggil fungsi dengan kumpulan catatan. Ini disebut pemetaan sumber peristiwa.

Saat Anda memanggil Fungsi Peristiwa, Anda dapat memilih untuk memanggilnya secara sinkron atau asinkron. Dengan invokasi sinkron, Anda menunggu fungsi untuk memproses peristiwa dan mengirimkan respons. Dengan invokasi asinkron, Lambda membuat antrean peristiwa untuk memproses dan mengembalikan respons dengan segera. Parameter `InvocationType` permintaan di API Invoke menentukan cara Lambda memanggil fungsi Anda. Nilai `RequestResponse` menunjukkan pemanggilan sinkron, dan nilai `Event` menunjukkan pemanggilan asinkron.

Untuk menjalankan Fungsi Acara Anda melalui IPv6, gunakan titik akhir tumpukan ganda publik Lambda. Dual-stack endpoint mendukung IPv4 dan IPv6. Titik akhir dual-stack Lambda menggunakan sintaks berikut:

```
protocol://lambda.us-east-1.api.aws
```

Anda juga dapat menggunakan URL fungsi Lambda untuk memanggil fungsi melalui IPv6. Titik akhir URL fungsi memiliki format berikut:

```
https://url-id.lambda-url.us-east-1.on.aws
```

Jika pemanggilan Fungsi Peristiwa menghasilkan kesalahan, untuk pemanggilan sinkron, lihat pesan kesalahan dalam respons dan coba lagi pemanggilan secara manual. Untuk pemanggilan asinkron, Lambda menangani percobaan ulang secara otomatis dan dapat mengirim catatan pemanggilan ke tujuan.

Memanggil Fungsi Acara Lambda secara sinkron

Saat Anda memanggil Fungsi Peristiwa Lambda secara sinkron, Lambda menjalankan fungsi dan menunggu respons. Ketika fungsi selesai, Lambda mengembalikan respons dari kode fungsi dengan data tambahan, seperti versi fungsi yang diaktifkan. Untuk menjalankan fungsi secara sinkron dengan AWS CLI, gunakan perintah. `invoke`

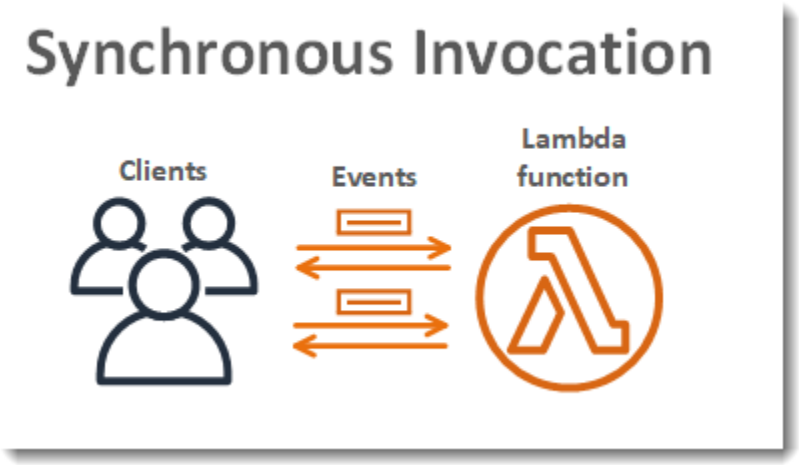
```
aws lambda invoke --function-name my-function \
    --cli-binary-format raw-in-base64-out \
    --payload '{ "key": "value" }' response.json
```

cli-binary-formatOpsi ini diperlukan jika Anda menggunakan AWS CLI versi 2. Untuk menjadikan ini pengaturan default, jalankan`aws configure set cli-binary-format raw-in-base64-out`. Untuk informasi selengkapnya, lihat opsi baris perintah global yang AWS CLI didukung di Panduan AWS Command Line Interface Pengguna untuk Versi 2.

Anda akan melihat output berikut:

```
{
    "ExecutedVersion": "$LATEST",
    "StatusCode": 200
}
```

Diagram berikut menunjukkan klien menjalankan Fungsi Peristiwa Lambda secara sinkron. Lambda mengirimkan peristiwa tersebut secara langsung ke fungsi dan mengirimkan respons fungsi kembali ke invoker.



`payload` adalah string yang berisi peristiwa dalam format JSON. Nama file tempat AWS CLI menulis respons dari fungsi adalah `response.json`. Jika Fungsi Acara mengembalikan objek atau kesalahan, badan respon adalah objek atau kesalahan dalam format JSON. Jika fungsi keluar tanpa kesalahan, badan responsnya adalah`null`.

> **Note**
>
> Lambda tidak menunggu ekstensi eksternal selesai sebelum mengirim respons. Ekstensi eksternal berjalan sebagai proses independen di lingkungan eksekusi dan terus berjalan setelah pemanggilan fungsi selesai. Untuk informasi selengkapnya, lihat Menambah Fungsi Acara Lambda menggunakan ekstensi Lambda.

Output dari perintah, yang ditampilkan di terminal, mencakup informasi dari header di respons dari Lambda. Ini termasuk versi yang memproses peristiwa (berguna saat Anda menggunakan alias), dan kode status yang dikembalikan oleh Lambda. Jika Lambda dapat menjalankan fungsi, kode statusnya adalah 200, meskipun fungsi tersebut mengembalikan kesalahan.

> **Note**
>
> Untuk Fungsi Acara dengan batas waktu yang lama, klien Anda mungkin terputus selama pemanggilan sinkron saat menunggu respons. Konfigurasikan klien HTTP, SDK, firewall, proksi, atau sistem operasi Anda untuk memungkinkan koneksi panjang dengan waktu habis atau pengaturan tetap aktif.

Jika Lambda tidak dapat menjalankan Fungsi Acara, kesalahan ditampilkan dalam output.

```
aws lambda invoke --function-name my-function \
    --cli-binary-format raw-in-base64-out \
    --payload value response.json
```

Anda akan melihat output berikut:

```
An error occurred (InvalidRequestContentException) when calling the Invoke operation:
 Could not parse request body into json: Unrecognized token 'value': was expecting
 ('true', 'false' or 'null')
 at [Source: (byte[])"value"; line: 1, column: 11]
```

AWS CLI Ini adalah alat sumber terbuka yang dapat Anda gunakan untuk berinteraksi dengan AWS layanan menggunakan perintah di shell baris perintah Anda. Untuk menyelesaikan langkah-langkah di bagian ini, Anda harus memiliki AWS CLI versi 2.

Anda dapat menggunakan AWS CLI untuk mengambil log untuk invokasi menggunakan opsi perintah `--log-type`. Respons berisi bidang `LogResult` yang memuat hingga 4 KB log berkode base64 dari invokasi.

Example mengambil ID log

Contoh berikut menunjukkan cara mengambil ID log dari `LogResult` untuk fungsi bernama `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

Anda akan melihat output berikut:

```
{
    "StatusCode": 200,
    "LogResult":
 "U1RBUlQgUmVxdWVzdElkOiA4N2QwNDRiOC1mMTU0LTExZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "ExecutedVersion": "$LATEST"
}
```

Example mendekode log

Pada prompt perintah yang sama, gunakan utilitas `base64` untuk mendekodekan log. Contoh berikut menunjukkan cara mengambil log berkode base64 untuk `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail \
--query 'LogResult' --output text --cli-binary-format raw-in-base64-out | base64 --
decode
```

cli-binary-formatOpsi ini diperlukan jika Anda menggunakan AWS CLI versi 2. Untuk menjadikan ini pengaturan default, jalankan`aws configure set cli-binary-format raw-in-base64-out`. Untuk informasi selengkapnya, lihat opsi baris perintah global yang AWS CLI didukung di Panduan AWS Command Line Interface Pengguna untuk Versi 2.

Anda akan melihat output berikut:

```
START RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Version: $LATEST
"AWS_SESSION_TOKEN": "AgoJb3JpZ2luX2VjELj...", "_X_AMZN_TRACE_ID": "Root=1-5d02e5ca-
f5792818b6fe8368e5b51d50;Parent=191db58857df8395;Sampled=0"",ask/lib:/opt/lib",
END RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8
REPORT RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8  Duration: 79.67 ms      Billed
 Duration: 80 ms         Memory Size: 128 MB     Max Memory Used: 73 MB
```

Utilitas `base64` tersedia di Linux, macOS, dan Ubuntu pada Windows. Pengguna macOS mungkin harus menggunakan `base64 -D`.

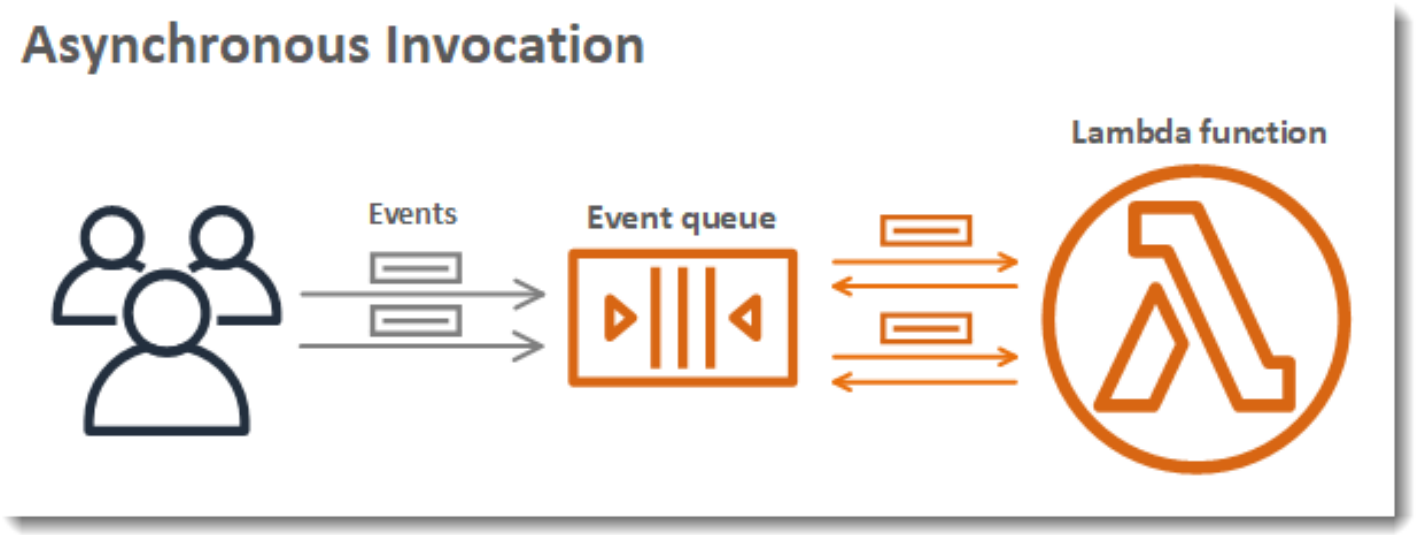
Untuk informasi selengkapnya tentang API `Invoke`, termasuk daftar lengkap parameter, header, dan kesalahan, lihat Panggil.

Saat Anda memanggil Fungsi Peristiwa secara langsung, Anda dapat memeriksa respons untuk kesalahan dan mencoba lagi. AWS SDK AWS CLI dan juga secara otomatis mencoba lagi pada batas waktu klien, pelambatan, dan kesalahan layanan. Lihat informasi yang lebih lengkap di Memahami perilaku coba lagi di Fungsi Peristiwa Lambda.

Memanggil Fungsi Peristiwa Lambda secara asinkron

Beberapa Layanan AWS, seperti Amazon Simple Storage Service (Amazon S3) Simple Storage Service (Amazon S3) dan Amazon Simple Notification Service (Amazon SNS), memanggil Fungsi Acara secara asinkron untuk memproses peristiwa. Anda juga dapat memanggil Fungsi Peristiwa Lambda secara asinkron menggunakan AWS CLI() atau salah AWS Command Line Interface satu SDK. AWS Saat Anda memanggil Fungsi Peristiwa secara asinkron, Anda tidak menunggu respons dari kode fungsi. Anda menyerahkan peristiwa ke Lambda dan Lambda menangani sisanya. Anda dapat mengonfigurasi cara Lambda menangani kesalahan, dan dapat mengirim catatan pemanggilan ke sumber daya hilir seperti Amazon Simple Queue Service (Amazon SQS) atau Amazon EventBridge () untuk menyatukan komponen aplikasi Anda. EventBridge

Diagram berikut menunjukkan klien yang mengaktifkan fungsi Lambda secara asinkron. Lambda membuat antrean peristiwa sebelum mengirimkannya ke fungsi.



Untuk invokasi asinkron, Lambda menempatkan peristiwa dalam antrean dan mengembalikan respons sukses informasi tambahan. Proses terpisah membaca peristiwa dari antrean dan mengirimnya ke fungsi Anda.

Untuk memanggil Fungsi Peristiwa Lambda secara asinkron menggunakan AWS Command Line Interface AWS CLI() atau salah satu SDK, setel parameter AWS ke. [InvocationType](#)`Event` Contoh berikut menunjukkan AWS CLI perintah untuk memanggil fungsi.

```
aws lambda invoke \
  --function-name my-function  \
  --invocation-type Event \
```

```
  --cli-binary-format raw-in-base64-out \
  --payload '{ "key": "value" }' response.json
```

Anda akan melihat output berikut:

```
{
    "StatusCode": 202
}
```

cli-binary-formatOpsi ini diperlukan jika Anda menggunakan AWS CLI versi 2. Untuk menjadikan ini pengaturan default, jalankan`aws configure set cli-binary-format raw-in-base64-out`. Untuk informasi selengkapnya, lihat opsi baris perintah global yang AWS CLI didukung di Panduan AWS Command Line Interface Pengguna untuk Versi 2.

File output (`response.json`) tidak berisi informasi apa pun, tetapi masih dibuat saat Anda menjalankan perintah ini. Jika Lambda tidak dapat menambahkan peristiwa ke antrean, pesan kesalahan akan muncul di output perintah.

Bagaimana Lambda menangani kesalahan dan mencoba ulang dengan pemanggilan asinkron

Lambda mengelola antrian peristiwa asinkron fungsi Anda dan mencoba lagi kesalahan. Jika fungsi mengembalikan kesalahan, secara default Lambda mencoba menjalankannya dua kali lagi, dengan menunggu satu menit antara dua upaya pertama, dan dua menit antara upaya kedua dan ketiga. Kesalahan Event Function mencakup kesalahan yang dikembalikan oleh kode fungsi dan kesalahan yang dikembalikan oleh runtime fungsi, seperti batas waktu.

Jika Fungsi Peristiwa tidak memiliki cukup konkurensi yang tersedia untuk memproses semua peristiwa, permintaan tambahan akan dibatasi. Untuk kesalahan pelambatan (429) dan kesalahan sistem (seri 500), Lambda mengembalikan acara ke antrian dan mencoba menjalankan fungsi lagi hingga 6 jam secara default. Interval percobaan lagi meningkat secara eksponensial dari 1 detik setelah percobaan pertama hingga maksimum 5 menit. Jika antrian berisi banyak entri, Lambda meningkatkan interval coba lagi dan mengurangi tingkat di mana ia membaca peristiwa dari antrian.

Bahkan jika Fungsi Acara Anda tidak menampilkan kesalahan, dimungkinkan untuk menerima peristiwa yang sama dari Lambda beberapa kali karena antrean itu sendiri pada akhirnya konsisten. Jika fungsi tidak dapat mengikuti peristiwa masuk, peristiwa juga dapat dihapus dari antrean tanpa

dikirim ke fungsi. Pastikan bahwa kode fungsi Anda secara rapi menangani duplikasi peristiwa, dan bahwa Anda memiliki cukup konkurensi tersedia untuk menangani semua invokasi.

Ketika antrean sangat panjang, acara baru mungkin menua sebelum Lambda memiliki kesempatan untuk mengirimnya ke Fungsi Acara Anda. Saat peristiwa kedaluwarsa atau gagal dalam semua upaya pemrosesan, Lambda menghapusnya. Anda dapat mengonfigurasi penanganan kesalahan untuk Fungsi Peristiwa untuk mengurangi jumlah percobaan ulang yang dilakukan Lambda, atau membuang peristiwa yang belum diproses dengan lebih cepat. Untuk menangkap peristiwa yang dibuang, konfigurasikan antrian huruf mati untuk fungsi tersebut. Untuk menangkap catatan pemanggilan yang gagal (seperti batas waktu atau kesalahan runtime), buat tujuan saat gagal.

Mengonfigurasi pengaturan penanganan kesalahan untuk pemanggilan asinkron Lambda

Gunakan pengaturan berikut untuk mengonfigurasi cara Lambda menangani kesalahan dan mencoba ulang untuk pemanggilan fungsi asinkron:

- MaximumEventAgeInSeconds: Jumlah waktu maksimum, dalam hitungan detik, Lambda menyimpan peristiwa dalam antrian acara asinkron sebelum membuangnya.
- MaximumRetryAttempts: Jumlah maksimum kali Lambda mencoba ulang peristiwa ketika fungsi mengembalikan kesalahan.

Gunakan konsol Lambda atau AWS CLI untuk mengonfigurasi pengaturan penanganan kesalahan pada Fungsi Acara, versi, atau alias.

Console

Untuk mengonfigurasi penanganan kesalahan

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih Fungsi Acara.
3. Pilih Konfigurasi, lalu pilihInvokasi asinkron.
4. Di bagian Invokasi asinkron, pilih Edit.
5. Konfigurasikan pengaturan berikut.
    - Usia maksimum peristiwa – Jumlah waktu maksimum yang dimiliki Lambda untuk menyimpan acara dalam antrean peristiwa asinkron, hingga 6 jam.

- Usaha percobaan ulang – Jumlah pengulangan Lambda saat fungsi mengembalikan kesalahan, antara 0 dan 2.

6. Pilih Simpan.

AWS CLI

Untuk mengonfigurasi pemanggilan asinkron dengan, gunakan perintah AWS CLI put-function-event-invoke-config. Contoh berikut mengkonfigurasi Fungsi Acara dengan usia acara maksimum 1 jam dan tidak ada percobaan ulang.

```
aws lambda put-function-event-invoke-config \
  --function-name error \
  --maximum-event-age-in-seconds 3600 \
  --maximum-retry-attempts 0
```

Perintah `put-function-event-invoke-config` menimpa konfigurasi fungsi, versi, atau alias yang ada. Untuk mengonfigurasi opsi tanpa mengatur ulang yang lain, gunakan update-function-event-invoke-config. Contoh berikut mengonfigurasi Lambda untuk mengirim catatan ke antrian SQS standar `destination` bernama saat peristiwa tidak dapat diproses.

```
aws lambda update-function-event-invoke-config \
  --function-name my-function \
  --destination-config '{"OnFailure":{"Destination": "arn:aws:sqs:us-
east-1:123456789012:destination"}}'
```

Anda akan melihat output berikut:

```
{
    "LastModified": 1573686021.479,
    "FunctionArn": "arn:aws:lambda:us-east-1:123456789012:function:my-function:
$LATEST",
    "MaximumRetryAttempts": 0,
    "MaximumEventAgeInSeconds": 3600,
    "DestinationConfig": {
        "OnSuccess": {},
        "OnFailure": {}
    }
}
```

Ketika peristiwa invokasi melebihi waktu maksimum atau gagal dalam semua upaya percobaan ulang, Lambda menghapusnya. Untuk menyimpan salinan peristiwa yang dibuang, konfigurasikan tujuan acara yang gagal.

Menangkap catatan pemanggilan asinkron Lambda

Lambda dapat mengirim catatan pemanggilan asinkron ke salah satu dari berikut ini. Layanan AWS

- Amazon SQS - Antrian SQS standar
- Amazon SNS - Topik SNS standar
- Amazon S3 - Bucket Amazon S3 (hanya gagal)
- AWS Lambda- Fungsi Lambda
- Amazon EventBridge - Bus EventBridge acara

Catatan invokasi berisi detail tentang permintaan dan respons dalam format JSON. Anda dapat mengonfigurasi tujuan terpisah untuk acara yang berhasil diproses, dan acara yang gagal dalam semua upaya pemrosesan. Atau, Anda dapat mengonfigurasi antrian Amazon SQS standar atau topik Amazon SNS standar sebagai antrian huruf mati untuk peristiwa yang dibuang. Untuk antrean dead-letter, Lambda hanya mengirimkan konten acara, tanpa perincian tentang responsnya.

Jika Lambda tidak dapat mengirim catatan ke tujuan yang telah Anda konfigurasikan, Lambda akan mengirimkan `DestinationDeliveryFailures` metrik ke Amazon. CloudWatch Hal ini dapat terjadi jika konfigurasi Anda menyertakan jenis tujuan yang tidak didukung, seperti antrean Amazon SQS FIFO atau topik Amazon SNS FIFO. Kesalahan pengiriman juga dapat terjadi karena kesalahan izin dan batas ukuran. Untuk informasi selengkapnya tentang metrik pemanggilan Lambda, lihat. the section called “Metrik invokasi”

> **Note**
>
> Untuk mencegah Fungsi Peristiwa memicu, Anda dapat mengatur konkurensi cadangan fungsi ke nol. Saat Anda menyetel konkurensi cadangan ke nol untuk fungsi yang dipanggil secara asinkron, Lambda mulai mengirim peristiwa baru ke antrian huruf mati yang dikonfigurasi atau tujuan peristiwa yang gagal, tanpa mencoba lagi. Untuk memproses peristiwa yang dikirim saat konkurensi cadangan disetel ke nol, Anda harus menggunakan peristiwa dari antrian surat mati atau tujuan acara yang gagal.

Menambahkan tujuan

Untuk menyimpan catatan pemanggilan asinkron, tambahkan tujuan ke Fungsi Acara Anda. Anda dapat memilih untuk mengirim pemanggilan yang berhasil atau gagal ke tujuan. Setiap fungsi dapat memiliki beberapa tujuan, sehingga Anda dapat mengonfigurasi tujuan terpisah untuk acara yang berhasil dan gagal. Setiap catatan yang dikirim ke tujuan adalah dokumen JSON dengan rincian tentang pemanggilan. Seperti pengaturan penanganan kesalahan, Anda dapat mengonfigurasi tujuan pada Fungsi Acara, versi fungsi, atau alias.

> **Tip**
>
> Anda juga dapat menyimpan catatan pemanggilan yang gagal untuk jenis pemetaan sumber peristiwa berikut: Amazon Kinesis, Amazon DynamoDB, dan Apache Kafka (Amazon MSK dan Apache Kafka yang dikelola sendiri).

Tabel berikut mencantumkan tujuan yang didukung untuk catatan pemanggilan asinkron. Agar Lambda berhasil mengirim catatan ke tujuan yang Anda pilih, pastikan bahwa peran eksekusi Fungsi Acara Anda juga berisi izin yang relevan. Tabel ini juga menjelaskan bagaimana setiap tipe tujuan menerima catatan pemanggilan JSON.

Jenis tujuan	Izin yang diperlukan	Destination-specific Format JSON
Antrean Amazon SQS	persegi: SendMessage	Lambda melewati catatan doa sebagai ke tujuan. `Message`
Topik Amazon SNS	SNS: Publikasikan	Lambda melewati catatan doa sebagai ke tujuan. `Message`
Bucket Amazon S3 (hanya pada kegagalan)	s3: PutObject s3: ListBucket	• Lambda menyimpan catatan pemanggilan sebagai objek JSON di bucket tujuan. • Nama objek S3 menggunak an konvensi penamaan berikut:

Jenis tujuan	Izin yang diperlukan	Destination-specific Format JSON
		`aws/lambda/async/<function-name>/YYYY/MM/DD/YYYY-MM-DDTHH.MM.SS-<Random UUID>`
Fungsi Acara Lambda	lambda: InvokeFunction	Lambda meneruskan catatan pemanggilan sebagai muatan ke fungsi.
EventBridge	peristiwa: PutEvents	• Lambda melewati catatan pemanggilan seperti dalam panggilan. `detail` PutEvents • Nilai untuk bidang `source` acara adalah`lambda`. • Nilai untuk bidang `detail-type` acara adalah “Hasil Pemanggilan Fungsi Lambda - Sukses” atau “Hasil Pemanggilan Fungsi Lambda - Kegagalan”. • Bidang `resource` acara berisi fungsi dan tujuan Amazon Resource Names (ARN). • Untuk bidang acara lainnya, lihat EventBridge Acara Amazon.

> **Note**
>
> Untuk tujuan Amazon S3, jika Anda telah mengaktifkan enkripsi pada bucket menggunakan kunci KMS, Fungsi Acara Anda juga memerlukan izin kms:. GenerateDataKey

> **Important**
>
> Saat menggunakan Amazon SNS sebagai tujuan, ketahuilah bahwa Amazon SNS memiliki batas ukuran pesan maksimum 256 KB. Jika payload pemanggilan asinkron Anda mendekati 1 MB, catatan pemanggilan (yang menyertakan muatan asli ditambah metadata tambahan) mungkin melebihi batas Amazon SNS dan menyebabkan kegagalan pengiriman. Pertimbangkan untuk menggunakan tujuan Amazon SQS atau Amazon S3 untuk muatan yang lebih besar.

Langkah-langkah berikut menjelaskan cara mengonfigurasi tujuan untuk suatu fungsi menggunakan konsol Lambda dan. AWS CLI

Console

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih Fungsi Acara.
3. Di bagian Gambaran umum fungsi, pilih Tambahkan tujuan.
4. Untuk Sumber, pilih Invokasi asinkron.
5. Untuk Kondisi, pilih dari opsi berikut:
   - Kegagalan – Kirimkan catatan saat peristiwa gagal dalam semua upaya pemrosesan atau melebihi waktu maksimum.
   - Sukses – Kirimkan catatan saat fungsi berhasil memproses invokasi asinkron.
6. Untuk Jenis tujuan, pilih jenis sumber daya yang menerima catatan invokasi.
7. Untuk Tujuan, pilih sumber daya.
8. Pilih Simpan.

AWS CLI

Untuk mengonfigurasi tujuan menggunakan AWS CLI, jalankan perintah update-function-event-invoke-config. Contoh berikut mengonfigurasi Lambda untuk mengirim catatan ke antrian SQS standar `destination` bernama saat peristiwa tidak dapat diproses.

```
aws lambda update-function-event-invoke-config \
  --function-name my-function \
  --destination-config '{"OnFailure":{"Destination": "arn:aws:sqs:us-
east-1:123456789012:destination"}}'
```

Praktik terbaik keamanan untuk tujuan Amazon S3

Menghapus bucket S3 yang dikonfigurasi sebagai tujuan tanpa menghapus tujuan dari konfigurasi fungsi Anda dapat menimbulkan risiko keamanan. Jika pengguna lain mengetahui nama bucket tujuan Anda, mereka dapat membuat ulang bucket di bucket tersebut. Akun AWS Catatan pemanggilan yang gagal akan dikirim ke bucket mereka, yang berpotensi mengekspos data dari fungsi Anda.

> **Warning**
>
> Untuk memastikan bahwa catatan pemanggilan dari Fungsi Peristiwa Anda tidak dapat dikirim ke bucket S3 di bucket lain Akun AWS, tambahkan kondisi ke peran eksekusi fungsi Anda yang membatasi `s3:PutObject` izin ke bucket di akun Anda.

Contoh berikut menunjukkan kebijakan IAM yang membatasi `s3:PutObject` izin fungsi Anda ke bucket di akun Anda. Kebijakan ini juga memberi Lambda `s3:ListBucket` izin yang dibutuhkan untuk menggunakan bucket S3 sebagai tujuan.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "S3BucketResourceAccountWrite",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:ListBucket"
            ],
```

```
            "Resource": [
                "arn:aws:s3:::*/*",
                "arn:aws:s3:::*"
            ],
            "Condition": {
                "StringEquals": {
                    "s3:ResourceAccount": "111122223333"
                }
            }
        }
    ]
}
```

Untuk menambahkan kebijakan izin ke peran eksekusi fungsi Anda menggunakan Konsol Manajemen AWS atau AWS CLI, lihat instruksi dalam prosedur berikut:

Console

Untuk menambahkan kebijakan izin ke peran eksekusi fungsi (konsol)

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih Lambda Event Function yang peran eksekusinya ingin Anda ubah.
3. Di tab Konfigurasi, pilih Izin.
4. Di tab Peran eksekusi, pilih nama Peran fungsi Anda untuk membuka halaman konsol IAM peran.
5. Tambahkan kebijakan izin ke peran dengan melakukan hal berikut:
    a. Di panel Kebijakan izin, pilih Tambahkan izin, lalu pilih Buat kebijakan sebaris.
    b. Di Editor kebijakan, pilih JSON.
    c. Rekatkan kebijakan yang ingin Anda tambahkan ke editor (ganti JSON yang ada), lalu pilih Berikutnya.
    d. Di bawah Detail kebijakan, masukkan nama Kebijakan.
    e. Pilih Buat kebijakan.

AWS CLI

Untuk menambahkan kebijakan izin ke peran eksekusi Fungsi Peristiwa (CLI)

1. Buat dokumen kebijakan JSON dengan izin yang diperlukan dan simpan di direktori lokal.

2. Gunakan perintah IAM `put-role-policy` CLI untuk menambahkan izin ke peran eksekusi fungsi Anda. Jalankan perintah berikut dari direktori tempat Anda menyimpan dokumen kebijakan JSON dan ganti nama peran, nama kebijakan, dan dokumen kebijakan dengan nilai Anda sendiri.

```
aws iam put-role-policy \
--role-name my_lambda_role \
--policy-name LambdaS3DestinationPolicy \
--policy-document file://my_policy.json
```

Contoh catatan pemanggilan

Ketika pemanggilan cocok dengan kondisi, Lambda mengirimkan dokumen JSON dengan rincian tentang pemanggilan ke tujuan. Contoh berikut menunjukkan catatan pemanggilan untuk peristiwa yang gagal dalam tiga upaya pemrosesan karena kesalahan Fungsi Peristiwa.

Example

```
{
    "version": "1.0",
    "timestamp": "2019-11-14T18:16:05.568Z",
    "requestContext": {
        "requestId": "e4b46cbf-b738-xmpl-8880-a18cdf61200e",
        "functionArn": "arn:aws:lambda:us-east-1:123456789012:function:my-function:
$LATEST",
        "condition": "RetriesExhausted",
        "approximateInvokeCount": 3
    },
    "requestPayload": {
        "ORDER_IDS": [
            "9e07af03-ce31-4ff3-xmpl-36dce652cb4f",
            "637de236-e7b2-464e-xmpl-baf57f86bb53",
            "a81ddca6-2c35-45c7-xmpl-c3a03a31ed15"
        ]
    },
    "responseContext": {
        "statusCode": 200,
        "executedVersion": "$LATEST",
        "functionError": "Unhandled"
    },
    "responsePayload": {
```

```
        "errorMessage": "RequestId: e4b46cbf-b738-xmpl-8880-a18cdf61200e Process exited
 before completing request"
    }
}
```

Catatan invokasi berisi perincian tentang peristiwa, respons, dan alasan mengapa catatan dikirim.

Menelusuri permintaan ke tujuan

Anda dapat menggunakan AWS X-Ray untuk melihat tampilan terhubung dari setiap permintaan saat antrian, diproses oleh Fungsi Peristiwa Lambda, dan diteruskan ke layanan tujuan. Saat Anda mengaktifkan X-Ray penelusuran untuk fungsi atau layanan yang memanggil fungsi, Lambda menambahkan X-Ray header ke permintaan dan meneruskan header ke layanan tujuan. Jejak dari layanan hulu secara otomatis ditautkan ke jejak dari Fungsi Acara Lambda hilir dan layanan tujuan, menciptakan tampilan ujung ke ujung dari seluruh aplikasi. Untuk informasi lebih lanjut tentang penelusuran, lihatVisualisasikan pemanggilan Fungsi Acara Lambda menggunakan AWS X-Ray.

Menambahkan antrian huruf mati

Sebagai alternatif untuk tujuan yang gagal, Anda dapat mengonfigurasi Fungsi Acara Anda dengan antrian huruf mati untuk menyimpan peristiwa yang dibuang untuk diproses lebih lanjut. Antrean dead-letter juga berfungsi sama seperti tujuan kegagalan dalam kondisi di mana ini digunakan saat suatu kejadian gagal dalam semua upaya pemrosesan atau kedaluwarsa tanpa diproses. Namun, Anda hanya dapat menambah atau menghapus antrian huruf mati di tingkat fungsi. Versi Fungsi Acara menggunakan pengaturan antrian huruf mati yang sama dengan versi yang tidak dipublikasikan ($LATEST). On-failure tujuan juga mendukung target tambahan dan menyertakan detail tentang respons fungsi dalam catatan pemanggilan.

Untuk memproses ulang peristiwa dalam antrian huruf mati, Anda dapat mengaturnya sebagai sumber peristiwa untuk Fungsi Peristiwa Lambda Anda. Atau, Anda dapat mengambil acara secara manual.

Anda dapat memilih antrian standar Amazon SQS atau topik standar Amazon SNS untuk antrian surat mati Anda. Antrian FIFO dan topik FIFO Amazon SNS tidak didukung.

- Antrean Amazon SQS – Antrean yang menahan kejadian yang gagal hingga terjadi. Pilih antrian standar Amazon SQS jika Anda mengharapkan satu entitas, seperti Fungsi Peristiwa Lambda atau CloudWatch alarm, untuk memproses peristiwa yang gagal. Untuk informasi selengkapnya, lihat Menggunakan Lambda dengan Amazon SQS.

- Topik Amazon SNS – Topik menyampaikan peristiwa yang gagal ke satu tujuan atau lebih. Pilih topik standar Amazon SNS jika Anda mengharapkan beberapa entitas bertindak atas peristiwa yang gagal. Misalnya, Anda dapat mengonfigurasi topik untuk mengirim peristiwa ke alamat email, Fungsi Peristiwa Lambda, titik akhir and/or HTTP. Untuk informasi selengkapnya, lihat Memanggil Fungsi Acara Lambda dengan notifikasi Amazon SNS.

Untuk mengirim acara ke antrian atau topik, Fungsi Acara Anda memerlukan izin tambahan. Tambahkan kebijakan dengan izin yang diperlukan ke peran eksekusi fungsi Anda. Jika antrean target atau topik dienkripsi dengan AWS KMS kunci yang dikelola pelanggan, pastikan peran eksekusi fungsi dan kebijakan berbasis sumber daya kunci berisi izin yang relevan.

Setelah membuat target dan memperbarui peran eksekusi Event Function Anda, tambahkan antrean huruf mati ke fungsi Anda. Anda dapat mengonfigurasi beberapa fungsi untuk mengirimkan peristiwa ke target yang sama.

Console

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih Fungsi Acara.
3. Pilih Konfigurasi, lalu pilihInvokasi asinkron.
4. Di bagian Invokasi asinkron, pilih Edit.
5. Setel layanan Dead-letter antrian ke Amazon SQS atau Amazon SNS.
6. Pilih antrean atau topik target.
7. Pilih Simpan.

AWS CLI

Untuk mengkonfigurasi antrian huruf mati dengan, gunakan perintah AWS CLI update-function-configuration.

```
aws lambda update-function-configuration \
  --function-name my-function \
  --dead-letter-config TargetArn=arn:aws:sns:us-east-1:123456789012:my-topic
```

Lambda mengirimkan peristiwa ke antrean surat mati sebagaimana adanya, dengan informasi tambahan dalam atribut. Anda dapat menggunakan informasi ini untuk mengidentifikasi kesalahan yang dikembalikan fungsi, atau untuk menghubungkan peristiwa dengan log atau jejak AWS X-Ray .

Dead-letter atribut pesan antrian

- RequestID (String) – ID permintaan invokasi. ID Permintaan muncul dalam log fungsi. Anda juga dapat menggunakan X-Ray SDK untuk merekam ID permintaan pada atribut dalam jejak. Anda kemudian dapat mencari jejak berdasarkan ID permintaan di X-Ray konsol.
- ErrorCode (Nomor) – Kode status HTTP.
- ErrorMessage (String) – 1 KB pertama dari pesan kesalahan.

Jika Lambda tidak dapat mengirim pesan ke antrean dead-letter, Lambda akan menghapus acara dan mengeluarkan metrik DeadLetterErrors. Ini dapat terjadi karena kurangnya izin, atau jika ukuran total pesan melebihi batasan untuk antrean atau topik target. Misalnya, katakan bahwa notifikasi Amazon SNS dengan ukuran tubuh mendekati 1 MB memicu fungsi yang menghasilkan kesalahan. Dalam hal ini, data peristiwa yang ditambahkan Amazon SNS, dikombinasikan dengan atribut yang ditambahkan Lambda, dapat menyebabkan pesan melebihi ukuran maksimum yang diizinkan dalam antrian huruf mati.

Jika Anda menggunakan Amazon SQS sebagai sumber peristiwa, konfigurasikan antrian huruf mati pada antrean Amazon SQS itu sendiri dan bukan pada Fungsi Peristiwa Lambda. Lihat informasi yang lebih lengkap di Menggunakan Lambda dengan Amazon SQS.

Bagaimana Lambda memproses catatan dari sumber acara berbasis streaming dan antrian

Pemetaan sumber peristiwa adalah sumber daya Lambda yang membaca item dari layanan berbasis streaming dan antrian dan memanggil fungsi dengan kumpulan catatan. Dalam pemetaan sumber peristiwa, sumber daya yang disebut poller acara secara aktif melakukan polling untuk pesan baru dan memanggil fungsi. Secara default, Lambda secara otomatis menskalakan poller peristiwa, tetapi untuk jenis sumber peristiwa tertentu, Anda dapat menggunakan mode yang disediakan untuk mengontrol jumlah minimum dan maksimum poller acara yang didedikasikan untuk pemetaan sumber acara Anda.

Layanan berikut menggunakan pemetaan sumber peristiwa untuk memanggil Fungsi Peristiwa Lambda:

- Amazon DocumentDB (dengan kompatibilitas MongoDB) (Amazon DocumentDB)
- Amazon DynamoDB
- Amazon Kinesis
- Amazon MQ
- Amazon Managed Streaming for Apache Kafka (Amazon MSK)
- Self-managed Apache Kafka
- Amazon Simple Queue Service (Amazon SQS)

> **Warning**
>
> Pemetaan sumber peristiwa Lambda memproses setiap peristiwa setidaknya sekali, dan pemrosesan duplikat catatan dapat terjadi. Untuk menghindari potensi masalah yang terkait dengan duplikat peristiwa, kami sangat menyarankan agar Anda membuat kode fungsi Anda idempoten. Untuk mempelajari lebih lanjut, lihat Bagaimana cara membuat fungsi Lambda saya idempoten di Pusat Pengetahuan. AWS

Bagaimana pemetaan sumber peristiwa berbeda dari pemicu langsung

Beberapa Layanan AWS dapat langsung memanggil Lambda Event Functions menggunakan pemicu. Layanan ini mendorong peristiwa ke Lambda, dan fungsi segera dipanggil saat peristiwa yang ditentukan terjadi. Pemicu cocok untuk peristiwa diskrit dan pemrosesan waktu nyata. Saat

Anda membuat pemicu menggunakan konsol Lambda, konsol berinteraksi dengan AWS layanan terkait untuk mengonfigurasi pemberitahuan peristiwa pada layanan tersebut. Pemicunya sebenarnya disimpan dan dikelola oleh layanan yang menghasilkan peristiwa, bukan oleh Lambda. Berikut adalah beberapa contoh layanan yang menggunakan pemicu untuk memanggil Lambda Event Functions:

- Amazon Simple Storage Service (Amazon S3): Memanggil fungsi saat objek dibuat, dihapus, atau dimodifikasi dalam bucket. Untuk informasi selengkapnya, lihat Tutorial: Menggunakan pemicu Amazon S3 untuk menjalankan Fungsi Peristiwa Lambda.
- Amazon Simple Notification Service (Amazon SNS): Memanggil fungsi saat pesan dipublikasikan ke topik SNS. Untuk informasi selengkapnya, lihat Tutorial: Menggunakan AWS Lambda dengan Layanan Pemberitahuan Sederhana Amazon.
- Amazon API Gateway: Memanggil fungsi saat permintaan API dibuat ke titik akhir tertentu. Untuk informasi selengkapnya, lihat Memanggil Fungsi Peristiwa Lambda menggunakan titik akhir Amazon API Gateway.

Pemetaan sumber peristiwa adalah sumber daya Lambda yang dibuat dan dikelola dalam layanan Lambda. Pemetaan sumber peristiwa dirancang untuk memproses data streaming volume tinggi atau pesan dari antrian. Memproses catatan dari aliran atau antrian dalam batch lebih efisien daripada memproses catatan secara individual.

Perilaku batching

Secara default, pemetaan sumber peristiwa mengumpulkan catatan bersama menjadi satu muatan yang dikirim Lambda ke Fungsi Acara Anda. Untuk menyempurnakan perilaku batching, Anda dapat mengonfigurasi jendela batching (MaximumBatchingWindowInSeconds) dan ukuran batch (). BatchSize Jendela batching adalah jumlah waktu maksimum untuk mengumpulkan catatan ke dalam satu muatan. Ukuran batch adalah jumlah maksimum catatan dalam satu batch. Lambda memanggil fungsi Anda ketika salah satu dari tiga kriteria berikut terpenuhi:

- Jendela batching mencapai nilai maksimumnya. Perilaku jendela batching default bervariasi tergantung pada sumber peristiwa tertentu.
  - Untuk sumber peristiwa Kinesis, DynamoDB, dan Amazon SQS: Jendela batching default adalah 0 detik. Ini berarti bahwa Lambda memanggil fungsi Anda segera setelah catatan tersedia. Untuk mengatur jendela batching, konfigurasikan`MaximumBatchingWindowInSeconds`. Anda dapat mengatur parameter ini ke nilai apa pun dari 0 hingga 300 detik dengan penambahan 1 detik. Jika Anda mengonfigurasi jendela batching, jendela berikutnya dimulai segera setelah pemanggilan fungsi sebelumnya selesai.

- Untuk Amazon MSK, sumber acara Apache Kafka, Amazon MQ, dan Amazon DocumentDB yang dikelola sendiri: Jendela batching default adalah 500 ms. Anda dapat mengonfigurasi `MaximumBatchingWindowInSeconds` ke nilai apa pun dari 0 detik hingga 300 detik dengan penambahan detik. Dalam mode yang disediakan untuk pemetaan sumber acara Kafka, saat Anda mengonfigurasi jendela batching, jendela berikutnya dimulai segera setelah batch sebelumnya selesai. Dalam pemetaan sumber peristiwa Kafka yang tidak disediakan, saat Anda mengonfigurasi jendela batching, jendela berikutnya dimulai segera setelah pemanggilan fungsi sebelumnya selesai.

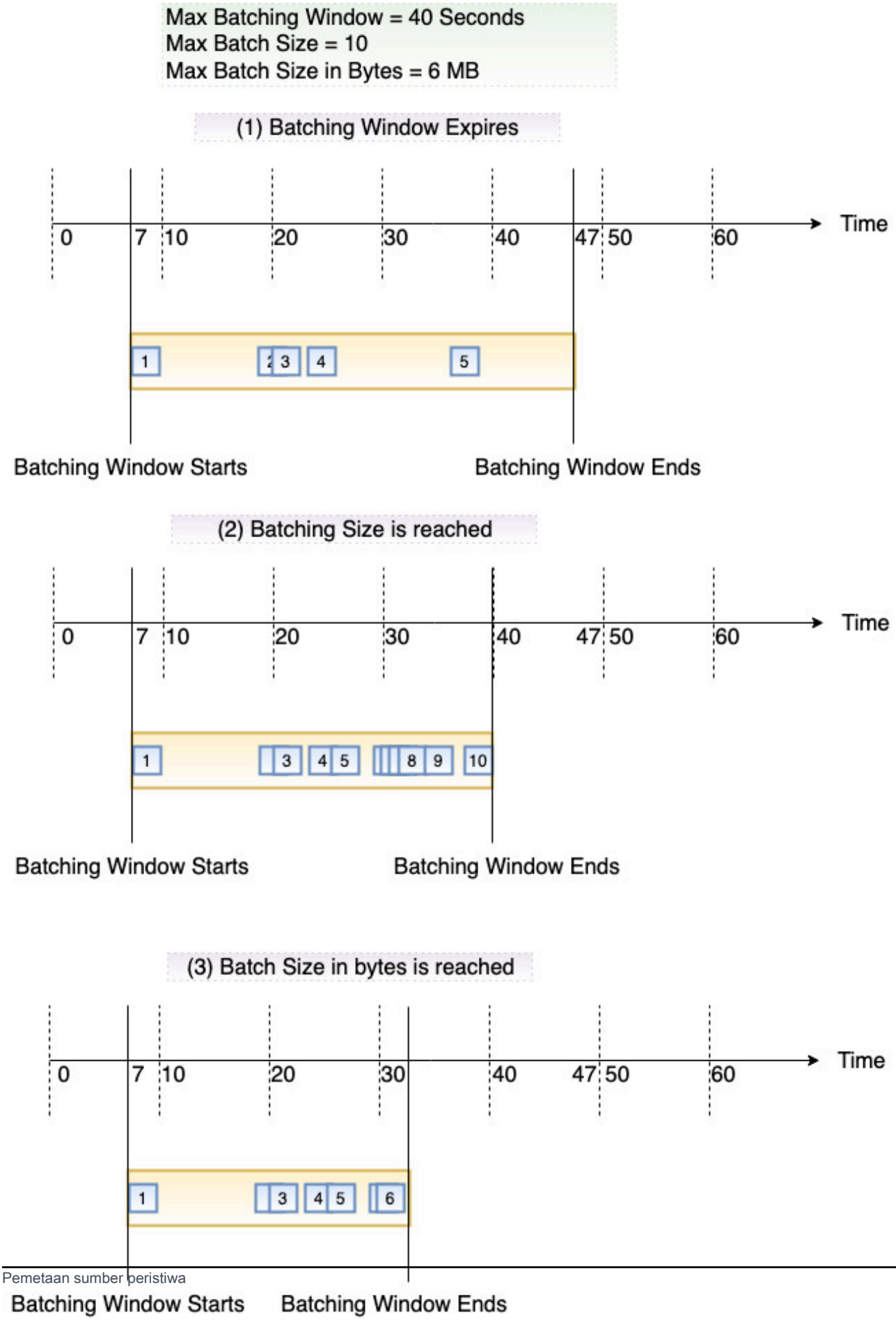
  Untuk meminimalkan latensi saat menggunakan pemetaan sumber peristiwa Kafka dalam mode yang disediakan, setel ke 0. `MaximumBatchingWindowInSeconds` Pengaturan ini memastikan bahwa Lambda mulai memproses batch berikutnya segera setelah menyelesaikan pemanggilan fungsi saat ini. Untuk informasi tambahan tentang pemrosesan latensi rendah, lihatLatensi rendah Apache Kafka.
- Untuk sumber acara Amazon MQ dan Amazon DocumentDB: Jendela batching default adalah 500 ms. Anda dapat mengonfigurasi `MaximumBatchingWindowInSeconds` ke nilai apa pun dari 0 detik hingga 300 detik dengan penambahan detik. Jendela batching dimulai segera setelah rekor pertama tiba.

  > **Note**
  >
  > Karena Anda hanya dapat mengubah `MaximumBatchingWindowInSeconds` dalam beberapa detik, Anda tidak dapat kembali ke jendela batching default 500 ms setelah Anda mengubahnya. Untuk mengembalikan jendela batching default, Anda harus membuat pemetaan sumber peristiwa baru.

- Ukuran batch terpenuhi. Ukuran batch minimum adalah 1. Ukuran batch default dan maksimum bergantung pada sumber acara. Untuk detail tentang nilai-nilai ini, lihat BatchSizespesifikasi untuk operasi `CreateEventSourceMapping` API.
- Ukuran payload mencapai 6 MB. Anda tidak dapat mengubah batas ini.

Diagram berikut menggambarkan ketiga kondisi ini. Misalkan jendela batching dimulai pada `t =` 7 detik. Dalam skenario pertama, jendela batching mencapai maksimum 40 detik dalam `t = 47` hitungan detik setelah mengumpulkan 5 catatan. Dalam skenario kedua, ukuran batch mencapai 10 sebelum jendela batching berakhir, sehingga jendela batching berakhir lebih awal. Dalam skenario

ketiga, ukuran muatan maksimum tercapai sebelum jendela batching berakhir, sehingga jendela batching berakhir lebih awal.


Max Batching Window = 40 Seconds
Max Batch Size = 10
Max Batch Size in Bytes = 6 MB
(1) Batching Window Expires
0
7
10
20
30
40
47
50
60
Time
1
3
4
5
Batching Window Starts
Batching Window Ends
(2) Batching Size is reached
0
7
10
20
30
40
47
50
60
Time
1
3
4
5
8
9
10
Batching Window Starts
Batching Window Ends
(3) Batch Size in bytes is reached
0
7
10
20
30
40
47
50
60
Time
1
3
4
5
6
Batching Window Starts
Batching Window Ends

Kami menyarankan Anda menguji dengan ukuran batch dan rekaman yang berbeda sehingga frekuensi polling dari setiap sumber acara disetel ke seberapa cepat Fungsi Acara Anda dapat menyelesaikan tugasnya. CreateEventSourceMapping`BatchSize`Parameter mengontrol jumlah maksimum catatan yang dapat dikirim ke fungsi Anda dengan setiap pemanggilan. Ukuran batch yang lebih besar sering kali dapat menyerap overhead secara lebih efisien di serangkaian catatan yang lebih besar, sehingga meningkatkan throughput Anda.

Lambda tidak menunggu ekstensi yang dikonfigurasi selesai sebelum mengirim batch berikutnya untuk diproses. Dengan kata lain, ekstensi Anda dapat terus berjalan saat Lambda memproses kumpulan catatan berikutnya. Hal ini dapat menyebabkan masalah pembatasan jika Anda melanggar pengaturan atau batasan konkurensi akun Anda. Untuk mendeteksi apakah ini merupakan masalah potensial, pantau fungsi Anda dan periksa apakah Anda melihat metrik konkurensi yang lebih tinggi dari yang diharapkan untuk pemetaan sumber peristiwa Anda. Karena waktu yang singkat di antara pemanggilan, Lambda mungkin secara singkat melaporkan penggunaan konkurensi yang lebih tinggi daripada jumlah pecahan. Ini bisa benar bahkan untuk Lambda Event Functions tanpa ekstensi.

Secara default, jika fungsi Anda mengembalikan kesalahan, pemetaan sumber peristiwa memproses ulang seluruh batch hingga fungsi berhasil, atau item dalam batch kedaluwarsa. Untuk memastikan pemrosesan dalam urutan, pemetaan sumber peristiwa menghentikan pemrosesan untuk pecahan yang terpengaruh hingga kesalahan teratasi. Untuk sumber aliran (DynamoDB dan Kinesis), Anda dapat mengonfigurasi jumlah maksimum kali Lambda mencoba ulang saat fungsi Anda mengembalikan kesalahan. Kesalahan layanan atau throttle di mana batch tidak mencapai fungsi Anda tidak diperhitungkan dalam upaya coba lagi. Anda juga dapat mengonfigurasi pemetaan sumber peristiwa untuk mengirim catatan pemanggilan ke tujuan saat membuang kumpulan peristiwa.

Mode yang disediakan

Pemetaan sumber peristiwa Lambda menggunakan poller acara untuk polling sumber acara Anda untuk pesan baru. Secara default, Lambda mengelola penskalaan otomatis poller ini berdasarkan volume pesan. Ketika lalu lintas pesan meningkat, Lambda secara otomatis meningkatkan jumlah poller acara untuk menangani beban, dan menguranginya saat lalu lintas menurun.

Dalam mode yang disediakan, Anda dapat menyempurnakan throughput pemetaan sumber acara Anda dengan menentukan batas minimum dan maksimum untuk sumber daya polling khusus yang tetap siap menangani pola lalu lintas yang diharapkan. Sumber daya ini secara otomatis menskalakan 3 kali lebih cepat untuk menangani lonjakan tiba-tiba dalam lalu lintas acara dan menyediakan kapasitas 16 kali lebih tinggi untuk memproses jutaan peristiwa. Ini membantu Anda

membangun beban kerja berbasis peristiwa yang sangat responsif dengan persyaratan performa yang ketat.

Di Lambda, poller peristiwa adalah unit komputasi dengan kemampuan throughput yang bervariasi menurut jenis sumber peristiwa. Untuk Amazon MSK dan Apache Kafka yang dikelola sendiri, setiap poller acara dapat menangani hingga 5 throughput atau hingga 5 MB/sec pemanggilan bersamaan. Misalnya, jika sumber acara Anda menghasilkan payload rata-rata 1 MB dan durasi rata-rata fungsi Anda adalah 1 detik, satu poller acara Kafka dapat mendukung 5 MB/sec throughput dan 5 pemanggilan Lambda bersamaan (dengan asumsi tidak ada transformasi payload). Untuk Amazon SQS, setiap poller peristiwa dapat menangani hingga 1 throughput atau hingga MB/sec 10 pemanggilan bersamaan. Menggunakan mode yang disediakan menimbulkan biaya tambahan berdasarkan penggunaan poller acara Anda. Untuk detail harganya, lihat Harga AWS Lambda.

Mode yang disediakan tersedia untuk Amazon MSK, Apache Kafka yang dikelola sendiri, dan sumber acara Amazon SQS. Meskipun pengaturan konkurensi memberi Anda kontrol atas penskalaan fungsi Anda, mode yang disediakan memberi Anda kontrol atas throughput pemetaan sumber peristiwa Anda. Untuk memastikan kinerja maksimum, Anda mungkin perlu menyesuaikan kedua pengaturan secara independen.

Mode yang disediakan sangat ideal untuk aplikasi real-time yang membutuhkan latensi pemrosesan peristiwa yang konsisten, seperti perusahaan jasa keuangan yang memproses umpan data pasar, platform e-commerce yang memberikan rekomendasi yang dipersonalisasi secara real-time, dan perusahaan game yang mengelola interaksi pemain langsung.

Setiap poller acara mendukung kapasitas throughput yang berbeda:

- Untuk Amazon MSK dan Apache Kafka yang dikelola sendiri: hingga 5 throughput atau hingga 5 MB/sec pemanggilan bersamaan
- Untuk Amazon SQS: hingga 1 throughput atau hingga 10 MB/sec pemanggilan bersamaan atau hingga 10 panggilan API polling SQS per detik.

Untuk pemetaan sumber peristiwa Amazon SQS, Anda dapat mengatur jumlah minimum poller antara 2 dan 200 dengan default 2, dan jumlah maksimum antara 2 dan 10.000 dengan default 200. Lambda menskalakan jumlah poller acara antara minimum dan maksimum yang dikonfigurasi, dengan cepat menambahkan hingga 1.000 konkurensi per menit untuk memberikan pemrosesan acara Anda yang konsisten dan latensi rendah.

Untuk pemetaan sumber acara Kafka, Anda dapat mengatur jumlah minimum poller antara 1 dan 200 dengan default 1, dan jumlah maksimum antara 1 dan 2.000 dengan default 200. Lambda

menskalakan jumlah poller acara antara minimum dan maksimum yang dikonfigurasi berdasarkan backlog acara Anda dalam topik Anda untuk memberikan pemrosesan latensi rendah dari acara Anda.

Perhatikan bahwa untuk sumber peristiwa Amazon SQS, setelan konkurensi maksimum tidak dapat digunakan dengan mode yang disediakan. Saat menggunakan mode yang disediakan, Anda mengontrol konkurensi melalui pengaturan poller peristiwa maksimum.

Untuk detail tentang mengonfigurasi mode yang disediakan, lihat bagian berikut:

- Mengonfigurasi mode yang disediakan untuk pemetaan sumber peristiwa MSK Amazon
- Mengkonfigurasi mode yang disediakan untuk pemetaan sumber acara Apache Kafka yang dikelola sendiri
- Menggunakan mode yang disediakan dengan pemetaan sumber peristiwa Amazon SQS

Untuk meminimalkan latensi dalam mode yang disediakan, atur `MaximumBatchingWindowInSeconds` ke 0. Pengaturan ini memastikan bahwa Lambda mulai memproses batch berikutnya segera setelah menyelesaikan pemanggilan fungsi saat ini. Untuk informasi tambahan tentang pemrosesan latensi rendah, lihatLatensi rendah Apache Kafka.

Setelah mengonfigurasi mode yang disediakan, Anda dapat mengamati penggunaan poller peristiwa untuk beban kerja Anda dengan memantau metrik. `ProvisionedPollers` Untuk informasi selengkapnya, lihat the section called “Metrik pemetaan sumber acara”.

API pemetaan sumber kejadian

Untuk mengelola sumber peristiwa dengan AWS Command Line Interface (AWS CLI) atau AWS SDK, Anda dapat menggunakan operasi API berikut:

- CreateEventSourceMapping
- ListEventSourceMappings
- GetEventSourceMapping
- UpdateEventSourceMapping
- DeleteEventSourceMapping

Menggunakan tag pada pemetaan sumber acara

Anda dapat menandai pemetaan sumber peristiwa untuk mengatur dan mengelola sumber daya Anda. Tag adalah pasangan nilai kunci bentuk bebas yang terkait dengan sumber daya Anda yang didukung secara keseluruhan. Layanan AWS Untuk informasi selengkapnya tentang kasus penggunaan tag, lihat Strategi penandaan umum di AWS Sumber Daya Penandaan dan Panduan Editor Tag.

Pemetaan sumber peristiwa dikaitkan dengan Fungsi Acara, yang dapat memiliki tag sendiri. Pemetaan sumber peristiwa tidak secara otomatis mewarisi tag dari fungsi. Anda dapat menggunakan AWS Lambda API untuk melihat dan memperbarui tag. Anda juga dapat melihat dan memperbarui tag saat mengelola pemetaan sumber peristiwa tertentu di konsol Lambda, termasuk yang menggunakan Mode Terketentuan untuk Amazon SQS.

Izin diperlukan untuk bekerja dengan tag

Untuk mengizinkan identitas AWS Identity and Access Management (IAM) (pengguna, grup, atau peran) membaca atau menyetel tag pada sumber daya, berikan izin yang sesuai:

- lambda: ListTags —Ketika sumber daya memiliki tag, berikan izin ini kepada siapa saja yang perlu memanggilnya`ListTags`. Untuk Fungsi Acara yang ditandai, izin ini juga diperlukan untuk`GetFunction`.
- lambda: TagResource —Berikan izin ini kepada siapa saja yang perlu memanggil `TagResource` atau melakukan tag saat membuat.

Secara opsional, pertimbangkan untuk memberikan UntagResource izin lambda: juga untuk mengizinkan `UntagResource` panggilan ke sumber daya.

Untuk informasi selengkapnya, lihat Identity-based Kebijakan IAM untuk Lambda.

Menggunakan tag dengan konsol Lambda

Anda dapat menggunakan konsol Lambda untuk membuat pemetaan sumber peristiwa yang memiliki tag, menambahkan tag ke pemetaan sumber peristiwa yang ada, dan memfilter pemetaan sumber peristiwa berdasarkan tag, termasuk yang dikonfigurasi dalam Mode Tertentu untuk Amazon SQS.

Saat Anda menambahkan pemicu untuk layanan berbasis streaming dan antrian yang didukung menggunakan konsol Lambda, Lambda secara otomatis membuat pemetaan sumber peristiwa. Untuk informasi lebih lanjut tentang sumber acara ini, lihatthe section called “Pemetaan sumber

peristiwa”. Untuk membuat pemetaan sumber peristiwa di konsol, Anda memerlukan prasyarat berikut:

- Fungsi Acara.
- Sumber peristiwa dari layanan yang terpengaruh.

Anda dapat menambahkan tag sebagai bagian dari antarmuka pengguna yang sama yang Anda gunakan untuk membuat atau memperbarui pemicu.

Untuk menambahkan tag saat Anda membuat pemetaan sumber peristiwa

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih nama Fungsi Acara Anda.
3. Di bagian Gambaran umum fungsi, pilih Tambah pemicu.
4. Di bawah konfigurasi Trigger, dalam daftar dropdown, pilih nama layanan sumber acara Anda berasal.
5. Berikan konfigurasi inti untuk sumber acara Anda. Untuk informasi selengkapnya tentang mengonfigurasi sumber acara Anda, lihat bagian untuk layanan terkait dithe section called “Mengintegrasikan layanan lain”.
6. Di bawah Konfigurasi pemetaan sumber peristiwa, pilih Pengaturan tambahan.
7. Di bawah Tag, pilih Tambahkan tag baru
8. Di bidang Key, masukkan kunci tag Anda. Untuk informasi tentang pembatasan penandaan, lihat Batas dan persyaratan penamaan tag di Panduan AWS Sumber Daya Penandaan dan Editor Tag.
9. Pilih Tambahkan.

Untuk menambahkan tag ke pemetaan sumber peristiwa yang ada

1. Buka pemetaan sumber Acara di konsol Lambda.
2. Dari daftar sumber daya, pilih UUID untuk pemetaan sumber peristiwa yang sesuai dengan ARN sumber Fungsi dan Acara Anda.
3. Dari daftar tab di bawah panel konfigurasi Umum, pilih Tag.
4. Pilih Kelola tanda.
5. Pilih Tambahkan tag baru.

6. Di bidang Key, masukkan kunci tag Anda. Untuk informasi tentang pembatasan penandaan, lihat Batas dan persyaratan penamaan tag di Panduan AWS Sumber Daya Penandaan dan Editor Tag.
7. Pilih Simpan.

Untuk memfilter pemetaan sumber peristiwa berdasarkan tag

1. Buka pemetaan sumber Acara di konsol Lambda.
2. Pilih kotak pencarian.
3. Dari daftar dropdown, pilih kunci tag Anda dari bawah subjudul Tag.
4. Pilih Gunakan: “tag-name” untuk melihat semua pemetaan sumber peristiwa yang ditandai dengan kunci ini, atau pilih Operator untuk memfilter lebih lanjut berdasarkan nilai.
5. Pilih nilai tag Anda untuk difilter dengan kombinasi kunci tag dan nilai.

Kotak pencarian juga mendukung pencarian kunci tag. Masukkan nama kunci untuk menemukannya dalam daftar.

Menggunakan tag dengan AWS CLI

Anda dapat menambahkan dan menghapus tag pada resource Lambda yang ada, termasuk pemetaan sumber peristiwa, dengan API Lambda. Anda juga dapat menambahkan tag saat membuat pemetaan sumber peristiwa untuk menjaga sumber daya diberi tag sepanjang siklus hidupnya.

Memperbarui tag dengan API tag Lambda

Anda dapat menambahkan dan menghapus tag untuk sumber daya Lambda yang didukung melalui operasi TagResourcedan UntagResourceAPI.

Anda dapat memanggil operasi ini menggunakan AWS CLI. Untuk menambahkan tag ke sumber daya yang ada, gunakan `tag-resource` perintah. Contoh ini menambahkan dua tag, satu dengan kunci *`Department`* dan satu dengan kunci*`CostCenter`*.

```
aws lambda tag-resource \
--resource arn:aws:lambda:us-east-2:123456789012:resource-type:my-resource \
--tags Department=Marketing,CostCenter=1234ABCD
```

Untuk menghapus tanda, gunakan perintah `untag-resource`. Contoh ini menghapus tag dengan kunci*`Department`*.

```
aws lambda untag-resource --resource arn:aws:lambda:us-east-1:123456789012:resource-type:resource-identifier \
--tag-keys Department
```

Menambahkan tag saat Anda membuat pemetaan sumber peristiwa

Untuk membuat pemetaan sumber peristiwa Lambda baru dengan tag, gunakan operasi API. CreateEventSourceMapping Tentukan parameter `Tags`. Anda dapat memanggil operasi ini dengan `create-event-source-mapping` AWS CLI perintah dan `--tags` opsi. Untuk informasi selengkapnya tentang perintah CLI, lihat create-event-source-mapping di Command Reference.AWS CLI

Sebelum menggunakan `Tags` parameter dengan`CreateEventSourceMapping`, pastikan bahwa peran Anda memiliki izin untuk menandai sumber daya bersama izin yang biasa diperlukan untuk operasi ini. Untuk informasi selengkapnya tentang izin untuk menandai, lihat. the section called “Izin diperlukan untuk bekerja dengan tag”

Melihat tag dengan API tag Lambda

Untuk melihat tag yang diterapkan ke sumber daya Lambda tertentu, gunakan operasi `ListTags` API. Untuk informasi selengkapnya, lihat ListTags.

Anda dapat memanggil operasi ini dengan `list-tags` AWS CLI perintah dengan memberikan ARN (Nama Sumber Daya Amazon).

```
aws lambda list-tags --resource arn:aws:lambda:us-east-1:123456789012:resource-type:resource-identifier
```

Memfilter sumber daya berdasarkan tag

Anda dapat menggunakan operasi AWS Resource Groups Tagging API GetResourcesAPI untuk memfilter sumber daya berdasarkan tag. `GetResources`Operasi menerima hingga 10 filter, dengan setiap filter berisi kunci tag dan hingga 10 nilai tag. Anda `GetResources` menyediakan `ResourceType` untuk memfilter berdasarkan jenis sumber daya tertentu.

Anda dapat memanggil operasi ini menggunakan `get-resources` AWS CLI perintah. Untuk contoh penggunaan`get-resources`, lihat mendapatkan sumber daya di Referensi Perintah AWS CLI.

Kontrol peristiwa mana yang dikirim Lambda ke Fungsi Acara Anda

Anda dapat menggunakan pemfilteran peristiwa untuk mengontrol rekaman mana dari aliran atau antrian yang dikirim Lambda ke fungsi Anda. Misalnya, Anda dapat menambahkan filter sehingga fungsi Anda hanya memproses pesan Amazon SQS yang berisi parameter data tertentu. Pemfilteran acara hanya berfungsi dengan pemetaan sumber peristiwa tertentu. Anda dapat menambahkan filter ke pemetaan sumber peristiwa untuk hal berikut: Layanan AWS

- Amazon DynamoDB
- Amazon Kinesis Data Streams
- Amazon MQ
- Amazon Managed Streaming for Apache Kafka (Amazon MSK)
- Self-managed Apache Kafka
- Amazon Simple Queue Service (Amazon SQS)

Untuk informasi spesifik tentang pemfilteran dengan sumber peristiwa tertentu, lihatthe section called “Menggunakan filter dengan berbeda Layanan AWS”. Lambda tidak mendukung pemfilteran acara untuk Amazon DocumentDB.

Secara default, Anda dapat menentukan hingga lima filter berbeda untuk pemetaan sumber peristiwa tunggal. Filter Anda secara logis disatukan. Jika rekaman dari sumber acara memenuhi satu atau beberapa filter Anda, Lambda menyertakan catatan di acara berikutnya yang dikirimkan ke fungsi Anda. Jika tidak ada filter Anda yang puas, Lambda membuang catatan tersebut.

> **Note**
>
> Jika Anda perlu menentukan lebih dari lima filter untuk sumber peristiwa, Anda dapat meminta peningkatan kuota hingga 10 filter untuk setiap sumber peristiwa. Jika Anda mencoba menambahkan lebih banyak filter daripada yang diizinkan kuota saat ini, Lambda mengembalikan kesalahan saat Anda mencoba membuat sumber acara.

Topik

- Memahami dasar-dasar penyaringan acara
- Menangani catatan yang tidak memenuhi kriteria filter
- Filter sintaks aturan

- Melampirkan kriteria filter ke pemetaan sumber peristiwa (konsol)
- Melampirkan kriteria filter ke pemetaan sumber peristiwa (AWS CLI)
- Melampirkan kriteria filter ke pemetaan sumber peristiwa (AWS SAM)
- Enkripsi kriteria filter
- Menggunakan filter dengan berbeda Layanan AWS

Memahami dasar-dasar penyaringan acara

Objek filter criteria (`FilterCriteria`) adalah struktur yang terdiri dari daftar filter (`Filters`). Setiap filter adalah struktur yang mendefinisikan pola penyaringan peristiwa ()`Pattern`. Pola adalah representasi string dari aturan filter JSON. Struktur suatu `FilterCriteria` objek adalah sebagai berikut.

```
{
   "Filters": [
        {
            "Pattern": "{ \"Metadata1\": [ rule1 ], \"data\": { \"Data1\":
 [ rule2 ] }}"
        }
    ]
}
```

Untuk kejelasan tambahan, berikut adalah nilai filter yang `Pattern` diperluas di JSON biasa.

```
{
    "Metadata1": [ rule1 ],
    "data": {
        "Data1": [ rule2 ]
    }
}
```

Pola filter Anda dapat mencakup properti metadata, properti data, atau keduanya. Parameter metadata yang tersedia dan format parameter data bervariasi sesuai dengan Layanan AWS yang bertindak sebagai sumber peristiwa. Misalnya, misalkan pemetaan sumber peristiwa Anda menerima catatan berikut dari antrean Amazon SQS:

```
{
    "messageId": "059f36b4-87a3-44ab-83d2-661975830a7d",
    "receiptHandle": "AQEBwJnKyrHigUMZj6rYigCgxlaS3SLy0a...",
```

```
    "body": "{\\n \"City\": \"Seattle\",\\n \"State\": \"WA\",\\n \"Temperature\":
 \"46\"\\n}",
    "attributes": {
        "ApproximateReceiveCount": "1",
        "SentTimestamp": "1545082649183",
        "SenderId": "AIDAIENQZJOLO23YVJ4VO",
        "ApproximateFirstReceiveTimestamp": "1545082649185"
    },
    "messageAttributes": {},
    "md5OfBody": "e4e68fb7bd0e697a0ae8f1bb342846b3",
    "eventSource": "aws:sqs",
    "eventSourceARN": "arn:aws:sqs:us-east-2:123456789012:my-queue",
    "awsRegion": "us-east-2"
}
```

- Properti metadata adalah bidang yang berisi informasi tentang peristiwa yang membuat catatan. Dalam contoh catatan Amazon SQS, properti metadata menyertakan bidang seperti`messageID`,, dan. `eventSourceArn` `awsRegion`
- Properti data adalah bidang catatan yang berisi data dari aliran atau antrian Anda. Dalam contoh peristiwa Amazon SQS, kunci untuk bidang data adalah`body`, dan properti data adalah bidang `CityState`, dan. `Temperature`

Berbagai jenis sumber acara menggunakan nilai kunci yang berbeda untuk bidang datanya. Untuk memfilter properti data, pastikan Anda menggunakan kunci yang benar dalam pola filter Anda. Untuk daftar kunci pemfilteran data, dan untuk melihat contoh pola filter untuk setiap yang didukung Layanan AWS, lihat. Menggunakan filter dengan berbeda Layanan AWS

Pemfilteran acara dapat menangani penyaringan JSON multi-level. Misalnya, pertimbangkan fragmen catatan berikut dari aliran DynamoDB:

```
"dynamodb": {
    "Keys": {
        "ID": {
            "S": "ABCD"
        }
        "Number": {
            "N": "1234"
    },
    ...
}
```

Misalkan Anda hanya ingin memproses catatan tersebut di mana nilai kunci sortir `Number` adalah 4567. Dalam hal ini, `FilterCriteria` objek Anda akan terlihat seperti ini:

```
{
    "Filters": [
        {
            "Pattern": "{ \"dynamodb\": { \"Keys\": { \"Number\": { \"N\":
 [ "4567" ] } } } }"
        }
    ]
}
```

Untuk kejelasan tambahan, berikut adalah nilai filter yang `Pattern` diperluas di JSON biasa.

```
{
    "dynamodb": {
        "Keys": {
            "Number": {
                "N": [ "4567" ]
                }
            }
        }
}
```

Menangani catatan yang tidak memenuhi kriteria filter

Cara Lambda menangani catatan yang tidak memenuhi kriteria filter Anda tergantung pada sumber acara.

- Untuk Amazon SQS, jika pesan tidak memenuhi kriteria filter Anda, Lambda secara otomatis menghapus pesan dari antrian. Anda tidak perlu menghapus pesan ini secara manual di Amazon SQS.
- Untuk Kinesis dan DynamoDB, setelah kriteria filter Anda mengevaluasi sebuah record, iterator stream akan melewati catatan ini. Jika catatan tidak memenuhi kriteria filter Anda, Anda tidak perlu menghapus catatan secara manual dari sumber acara Anda. Setelah periode retensi, Kinesis dan DynamoDB secara otomatis menghapus catatan lama ini. Jika Anda ingin catatan dihapus lebih cepat, lihat Mengubah Periode Retensi Data.
- Untuk Amazon MSK, Apache Kafka yang dikelola sendiri, dan pesan Amazon MQ, Lambda menghapus pesan yang tidak cocok dengan semua bidang yang disertakan dalam filter. Untuk

Amazon MSK dan Apache Kafka yang dikelola sendiri, Lambda melakukan offset untuk pesan yang cocok dan tak tertandingi setelah berhasil menjalankan Fungsi Acara. Untuk Amazon MQ, Lambda mengakui pesan yang cocok setelah berhasil menjalankan fungsi, dan mengakui pesan yang tak tertandingi saat memfilternya.

Filter sintaks aturan

Untuk aturan filter, Lambda mendukung EventBridge aturan Amazon dan menggunakan sintaks yang sama seperti. EventBridge Untuk informasi selengkapnya, lihat pola EventBridge acara Amazon di Panduan EventBridge Pengguna Amazon.

Berikut ini adalah ringkasan dari semua operator perbandingan yang tersedia untuk pemfilteran acara Lambda.

Operator perbandingan	Contoh	Sintaks aturan
Kosong	UserID adalah kosong	“userId”: [null]
Kosong	LastName kosong	"LastName": [""]
Sama dengan	Namanya adalah “Alice”	“Nama”: [“Alice”]
Sama dengan (abaikan kasus)	Namanya adalah “Alice”	“Name”: [{“equals-ignore-case”: “alice”}]
Dan	Lokasi adalah “New York” dan Hari adalah “Senin”	“Lokasi”: [“New York”], “Hari”: ["Senin"]
Atau	PaymentType adalah “Kredit” atau “Debit”	“PaymentType“: [“Kredit”, “Debit"]
Atau (beberapa bidang)	Lokasi adalah “New York”, atau Hari adalah “Senin”.	“$or”: [{“Lokasi”: [“New York”]}, {“Hari”: [“Senin”]}]
Bukan	Cuaca hanyalah “Hujan”	“Weather”: [{“anything-but”: [“Hujan”]}]
Numerik (sama dengan)	Harga 100	“Harga”: [{“numerik”: [“=”, 100]}]

Operator perbandingan	Contoh	Sintaks aturan
Numerik (rentang)	Harga lebih dari 10, dan kurang dari atau sama dengan 20	“Harga”: [{“numerik”: [“>”, 10, “<=”, 20]}]
Exists	ProductName ada	“ProductName“: [{“exists”: true}]
Tidak ada	ProductName tidak ada	“ProductName“: [{“exists”: false}]
Dimulai dengan	Wilayah ada di AS	“Region”: [{"prefix”: “us-”}]
Ends with	FileName diakhiri dengan ekstensi.png.	“FileName“: [{“akhiran”: “.png”}]

> **Note**
>
> Seperti EventBridge, untuk string, Lambda menggunakan pencocokan karakter demi karakter yang tepat tanpa pelipatan kasus atau normalisasi string lainnya. Untuk angka, Lambda juga menggunakan representasi string. Sebagai contoh, 300, 300,0, dan 3,0e2 dianggap tidak sama.

Perhatikan bahwa operator Exists hanya bekerja pada node daun di JSON sumber acara Anda. Itu tidak cocok dengan node perantara. Misalnya, dengan JSON berikut, pola filter tidak `{ "person": { "address": [ { "exists": true } ] } }` akan menemukan kecocokan karena `"address"` merupakan simpul perantara.

```
{
  "person": {
    "name": "John Doe",
    "age": 30,
    "address": {
      "street": "123 Main St",
      "city": "Anytown",
      "country": "USA"
    }
```

```
  }
}
```

Melampirkan kriteria filter ke pemetaan sumber peristiwa (konsol)

Ikuti langkah-langkah ini untuk membuat pemetaan sumber peristiwa baru dengan kriteria filter menggunakan konsol Lambda.

Untuk membuat pemetaan sumber acara baru dengan kriteria filter (konsol)

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih nama Fungsi Acara untuk membuat pemetaan sumber peristiwa.
3. Di bagian Gambaran umum fungsi, pilih Tambah pemicu.
4. Untuk konfigurasi Trigger, pilih jenis pemicu yang mendukung pemfilteran peristiwa. Untuk daftar layanan yang didukung, lihat daftar di awal halaman ini.
5. Perluas Pengaturan tambahan.
6. Di bawah Kriteria filter, pilih Tambah, lalu tentukan dan masukkan filter Anda. Misalnya, Anda dapat memasukkan yang berikut ini.

   ```
   { "Metadata" : [ 1, 2 ] }
   ```

   Ini menginstruksikan Lambda untuk memproses hanya catatan di mana `Metadata` bidang sama dengan 1 atau 2. Anda dapat terus memilih Tambah untuk menambahkan lebih banyak filter hingga jumlah maksimum yang diizinkan.
7. Setelah selesai menambahkan filter, pilih Simpan.

Saat Anda memasukkan kriteria filter menggunakan konsol, Anda hanya memasukkan pola filter dan tidak perlu memberikan tanda kutip `Pattern` kunci atau escape. Pada langkah 6 dari instruksi sebelumnya, `{ "Metadata" : [ 1, 2 ] }` sesuai dengan yang berikut ini. `FilterCriteria`

```
{
   "Filters": [
      {
          "Pattern": "{ \"Metadata\" : [ 1, 2 ] }"
      }
   ]
}
```

Setelah membuat pemetaan sumber acara di konsol, Anda dapat melihat format `FilterCriteria` dalam detail pemicu. Untuk lebih banyak contoh pembuatan filter acara menggunakan konsol, lihatMenggunakan filter dengan berbeda Layanan AWS.

Melampirkan kriteria filter ke pemetaan sumber peristiwa (AWS CLI)

Misalkan Anda ingin pemetaan sumber peristiwa memiliki yang berikut: `FilterCriteria`

```
{
   "Filters": [
      {
          "Pattern": "{ \"Metadata\" : [ 1, 2 ] }"
      }
   ]
}
```

Untuk membuat pemetaan sumber peristiwa baru dengan kriteria filter ini menggunakan AWS Command Line Interface (AWS CLI), jalankan perintah berikut.

```
aws lambda create-event-source-mapping \
    --function-name my-function \
    --event-source-arn arn:aws:sqs:us-east-2:123456789012:my-queue \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"Metadata\" : [ 1, 2 ]}"}]}'
```

Perintah create-event-source-mapping ini membuat pemetaan sumber peristiwa Amazon SQS baru untuk fungsi dengan yang ditentukan. `my-function` `FilterCriteria`

Untuk menambahkan kriteria filter ini ke pemetaan sumber peristiwa yang ada, jalankan perintah berikut.

```
aws lambda update-event-source-mapping \
    --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"Metadata\" : [ 1, 2 ]}"}]}'
```

Perhatikan bahwa untuk memperbarui pemetaan sumber peristiwa, Anda memerlukan UUID-nya. Anda bisa mendapatkan UUID dari panggilan list-event-source-mappings. Lambda juga mengembalikan UUID dalam respons CLI create-event-source-mapping.

Untuk menghapus kriteria filter dari sumber peristiwa, Anda dapat menjalankan perintah update-event-source-mapping berikut dengan objek kosong. `FilterCriteria`

```
aws lambda update-event-source-mapping \
    --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
    --filter-criteria "{}"
```

Untuk lebih banyak contoh pembuatan filter acara menggunakan AWS CLI, lihatMenggunakan filter dengan berbeda Layanan AWS.

Melampirkan kriteria filter ke pemetaan sumber peristiwa (AWS SAM)

Misalkan Anda ingin mengonfigurasi sumber acara AWS SAM untuk menggunakan kriteria filter berikut:

```
{
   "Filters": [
      {
          "Pattern": "{ \"Metadata\" : [ 1, 2 ] }"
      }
   ]
}
```

Untuk menambahkan kriteria filter ini ke pemetaan sumber peristiwa Anda, masukkan cuplikan berikut ke dalam template YAMM untuk sumber acara Anda.

```
FilterCriteria:
  Filters:
    - Pattern: '{"Metadata": [1, 2]}'
```

Untuk informasi selengkapnya tentang membuat dan mengonfigurasi AWS SAM templat untuk pemetaan sumber peristiwa, lihat EventSourcebagian Panduan AWS SAM Pengembang. Untuk lebih banyak contoh pembuatan filter acara menggunakan AWS SAM templat, lihatMenggunakan filter dengan berbeda Layanan AWS.

Enkripsi kriteria filter

Secara default, Lambda tidak mengenkripsi objek kriteria filter Anda. Untuk kasus penggunaan di mana Anda mungkin menyertakan informasi sensitif dalam objek kriteria filter Anda, Anda dapat menggunakan kunci KMS Anda sendiri untuk mengenkripsinya.

Setelah mengenkripsi objek kriteria filter, Anda dapat melihat versi teks biasa menggunakan panggilan API. GetEventSourceMapping Anda harus memiliki `kms:Decrypt` izin untuk dapat berhasil melihat kriteria filter dalam teks biasa.

> **Note**
>
> Jika objek kriteria filter Anda dienkripsi, Lambda menyunting nilai `FilterCriteria` bidang dalam respons panggilan. ListEventSourceMappings Sebaliknya, bidang ini ditampilkan sebagai`null`. Untuk melihat nilai sebenarnya dari`FilterCriteria`, gunakan GetEventSourceMappingAPI.
> Untuk melihat nilai yang didekripsi `FilterCriteria` di konsol, pastikan peran IAM Anda berisi izin untuk. GetEventSourceMapping

Anda dapat menentukan kunci KMS Anda sendiri melalui konsol, API/CLI, atau CloudFormation.

Untuk mengenkripsi kriteria filter dengan kunci KMS milik pelanggan (konsol)

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih Tambahkan pemicu. Jika Anda sudah memiliki pemicu yang ada, pilih tab Konfigurasi, lalu pilih Pemicu. Pilih pemicu yang ada, dan pilih Edit.
3. Pilih kotak centang di sebelah Enkripsi dengan kunci KMS yang dikelola pelanggan.
4. Untuk Pilih kunci enkripsi KMS yang dikelola pelanggan, pilih kunci aktif yang ada atau buat kunci baru. Bergantung pada operasinya, Anda memerlukan beberapa atau semua izin berikut:`kms:DescribeKey`,`kms:GenerateDataKey`, dan`kms:Decrypt`. Gunakan kebijakan kunci KMS untuk memberikan izin ini.

Jika Anda menggunakan kunci KMS Anda sendiri, operasi API berikut harus diizinkan dalam kebijakan kunci:

- `kms:Decrypt`— Harus diberikan kepada kepala layanan Lambda regional ()`lambda.`*`AWS_region`*`.amazonaws.com`. Ini memungkinkan Lambda untuk mendekripsi data dengan kunci KMS ini.
  - Untuk mencegah masalah wakil lintas layanan yang membingungkan, kebijakan kunci menggunakan kunci kondisi `aws:SourceArn`global. Nilai `aws:SourceArn` kunci yang benar adalah ARN sumber pemetaan sumber peristiwa Anda, sehingga Anda dapat menambahkan ini ke kebijakan Anda hanya setelah Anda mengetahui ARN-nya. Lambda juga meneruskan

`aws:lambda:EventSourceArn` kunci `aws:lambda:FunctionArn` dan dan nilainya masing-masing dalam konteks enkripsi saat membuat permintaan dekripsi ke KMS. Nilai-nilai ini harus sesuai dengan kondisi yang ditentukan dalam kebijakan kunci agar permintaan dekripsi berhasil. Anda tidak perlu menyertakan sumber EventSourceArn acara Self-managed Kafka karena mereka tidak memiliki EventSourceArn.

- `kms:Decrypt`— Juga harus diberikan kepada prinsipal yang bermaksud menggunakan kunci untuk melihat kriteria filter teks biasa dalam GetEventSourceMappingatau DeleteEventSourceMappingpanggilan API.
- `kms:DescribeKey`— Memberikan rincian kunci yang dikelola pelanggan untuk memungkinkan prinsipal yang ditentukan untuk menggunakan kunci.
- `kms:GenerateDataKey`— Memberikan izin bagi Lambda untuk menghasilkan kunci data untuk mengenkripsi kriteria filter, atas nama prinsipal yang ditentukan (enkripsi amplop).

Anda dapat menggunakan AWS CloudTrail untuk melacak AWS KMS permintaan yang dibuat Lambda atas nama Anda. Untuk contoh CloudTrail peristiwa, lihat???.

Kami juga merekomendasikan menggunakan tombol `kms:ViaService`kondisi untuk membatasi penggunaan kunci KMS hanya untuk permintaan dari Lambda. Nilai kunci ini adalah kepala layanan Lambda regional ()`lambda.`*`AWS_region`*`.amazonaws.com`. Berikut ini adalah contoh kebijakan kunci yang memberikan semua izin yang relevan:

Example AWS KMS kebijakan kunci

JSON

```
{
    "Version":"2012-10-17",
    "Id": "example-key-policy-1",
    "Statement": [
        {
            "Sid": "Allow Lambda to decrypt using the key",
            "Effect": "Allow",
            "Principal": {
                "Service": "lambda.us-east-1.amazonaws.com"
            },
            "Action": [
                "kms:Decrypt"
            ],
            "Resource": "*",
```

```
            "Condition": {
                "ArnEquals" : {
                    "aws:SourceArn": [
                        "arn:aws:lambda:us-east-1:123456789012:event-source-
mapping:<esm_uuid>"
                    ]
                },
                "StringEquals": {

 "kms:EncryptionContext:aws:lambda:FunctionArn": "arn:aws:lambda:us-
east-1:123456789012:function:test-function",

 "kms:EncryptionContext:aws:lambda:EventSourceArn": "arn:aws:sqs:us-
east-1:123456789012:test-queue"
                }
            }
        },
        {
            "Sid": "Allow actions by an AWS account on the key",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::123456789012:root"
            },
            "Action": "kms:*",
            "Resource": "*"
        },
        {
            "Sid": "Allow use of the key to specific roles",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::123456789012:role/ExampleRole"
            },
            "Action": [
                "kms:Decrypt",
                "kms:DescribeKey",
                "kms:GenerateDataKey"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals" : {
                    "kms:ViaService": "lambda.us-east-1.amazonaws.com"
                }
            }
        }
```

```
    ]
}
```

Untuk menggunakan kunci KMS Anda sendiri untuk mengenkripsi kriteria filter, Anda juga dapat menggunakan perintah berikut CreateEventSourceMapping AWS CLI . Tentukan ARN kunci KMS dengan `--kms-key-arn` bendera.

```
aws lambda create-event-source-mapping --function-name my-function \
    --maximum-batching-window-in-seconds 60 \
    --event-source-arn arn:aws:sqs:us-east-1:123456789012:my-queue \
    --filter-criteria "{\"filters\": [{\"pattern\": \"{\"a\": [\"1\", \"2\"]}\" }]}" \
    --kms-key-arn arn:aws:kms:us-east-1:123456789012:key/055efbb4-xmpl-4336-
ba9c-538c7d31f599
```

Jika Anda memiliki pemetaan sumber peristiwa yang ada, gunakan UpdateEventSourceMapping AWS CLI perintah sebagai gantinya. Tentukan ARN kunci KMS dengan `--kms-key-arn` bendera.

```
aws lambda update-event-source-mapping --function-name my-function \
    --maximum-batching-window-in-seconds 60 \
    --event-source-arn arn:aws:sqs:us-east-1:123456789012:my-queue \
    --filter-criteria "{\"filters\": [{\"pattern\": \"{\"a\": [\"1\", \"2\"]}\" }]}" \
    --kms-key-arn arn:aws:kms:us-east-1:123456789012:key/055efbb4-xmpl-4336-
ba9c-538c7d31f599
```

Operasi ini menimpa setiap kunci KMS yang sebelumnya ditentukan. Jika Anda menentukan `--kms-key-arn` bendera bersama dengan argumen kosong, Lambda berhenti menggunakan kunci KMS Anda untuk mengenkripsi kriteria filter. Sebaliknya, Lambda default kembali menggunakan kunci. Amazon-owned

Untuk menentukan kunci KMS Anda sendiri dalam CloudFormation template, gunakan `KMSKeyArn` properti jenis `AWS::Lambda::EventSourceMapping` sumber daya. Misalnya, Anda dapat menyisipkan cuplikan berikut ke dalam template YAMB untuk sumber acara Anda.

```
MyEventSourceMapping:
  Type: AWS::Lambda::EventSourceMapping
  Properties:
    ...
    FilterCriteria:
      Filters:
        - Pattern: '{"a": [1, 2]}'
```

```
    KMSKeyArn: "arn:aws:kms:us-east-1:123456789012:key/055efbb4-xmpl-4336-
ba9c-538c7d31f599"
    ...
```

Untuk dapat melihat kriteria filter terenkripsi Anda dalam teks biasa dalam panggilan GetEventSourceMappingatau DeleteEventSourceMappingAPI, Anda harus memiliki izin. `kms:Decrypt`

Mulai 6 Agustus 2024, `FilterCriteria` bidang tidak lagi muncul di AWS CloudTrail log dari CreateEventSourceMappingUpdateEventSourceMapping, dan panggilan DeleteEventSourceMappingAPI jika fungsi Anda tidak menggunakan pemfilteran peristiwa. Jika fungsi Anda menggunakan pemfilteran peristiwa, `FilterCriteria` bidang akan muncul sebagai empty (`{}`). Anda masih dapat melihat kriteria filter Anda dalam teks biasa dalam respons panggilan GetEventSourceMappingAPI jika Anda memiliki `kms:Decrypt` izin untuk kunci KMS yang benar.

Contoh entri CloudTrail log untuk Create/Update/DeleteEventSourceMapping panggilan

Dalam AWS CloudTrail contoh entri log berikut untuk CreateEventSourceMapping panggilan, `FilterCriteria` muncul sebagai empty (`{}`) karena fungsi menggunakan pemfilteran peristiwa. Ini adalah kasus bahkan jika `FilterCriteria` objek berisi kriteria filter yang valid yang fungsi Anda gunakan secara aktif. Jika fungsi tidak menggunakan pemfilteran peristiwa, tidak CloudTrail akan menampilkan `FilterCriteria` bidang sama sekali dalam entri log.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROA123456789EXAMPLE:userid1",
        "arn": "arn:aws:sts::123456789012:assumed-role/Example/example-role",
        "accountId": "123456789012",
        "accessKeyId": "ASIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROA987654321EXAMPLE",
                "arn": "arn:aws:iam::123456789012:role/User1",
                "accountId": "123456789012",
                "userName": "User1"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2024-05-09T20:35:01Z",
```

```
                "mfaAuthenticated": "false"
            }
        },
        "invokedBy": "AWS Internal"
    },
    "eventTime": "2024-05-09T21:05:41Z",
    "eventSource": "lambda.amazonaws.com",
    "eventName": "CreateEventSourceMapping20150331",
    "awsRegion": "us-east-2",
    "sourceIPAddress": "AWS Internal",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "eventSourceArn": "arn:aws:sqs:us-east-2:123456789012:example-queue",
        "functionName": "example-function",
        "enabled": true,
        "batchSize": 10,
        "filterCriteria": {},
        "kMSKeyArn": "arn:aws:kms:us-east-2:123456789012:key/a1b2c3d4-5678-90ab-cdef-
EXAMPLE11111",
        "scalingConfig": {},
        "maximumBatchingWindowInSeconds": 0,
        "sourceAccessConfigurations": []
    },
    "responseElements": {
        "uUID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEaaaaa",
        "batchSize": 10,
        "maximumBatchingWindowInSeconds": 0,
        "eventSourceArn": "arn:aws:sqs:us-east-2:123456789012:example-queue",
        "filterCriteria": {},
        "kMSKeyArn": "arn:aws:kms:us-east-2:123456789012:key/a1b2c3d4-5678-90ab-cdef-
EXAMPLE11111",
        "functionArn": "arn:aws:lambda:us-east-2:123456789012:function:example-
function",
        "lastModified": "May 9, 2024, 9:05:41 PM",
        "state": "Creating",
        "stateTransitionReason": "USER_INITIATED",
        "functionResponseTypes": [],
        "eventSourceMappingArn": "arn:aws:lambda:us-east-2:123456789012:event-source-
mapping:a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbb"
    },
    "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE33333",
    "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
    "readOnly": false,
    "eventType": "AwsApiCall",
```

```
    "managementEvent": true,
    "recipientAccountId": "123456789012",
    "eventCategory": "Management",
    "sessionCredentialFromConsole": "true"
}
```

Menggunakan filter dengan berbeda Layanan AWS

Berbagai jenis sumber acara menggunakan nilai kunci yang berbeda untuk bidang datanya. Untuk memfilter properti data, pastikan Anda menggunakan kunci yang benar dalam pola filter Anda. Tabel berikut memberikan kunci penyaringan untuk setiap didukung Layanan AWS.

Layanan AWS	Kunci penyaringan
DynamoDB	`dynamodb`
Kinesis	`data`
Amazon MQ	`data`
Amazon MSK	`value`
Self-managed Apache Kafka	`value`
Amazon SQS	`body`

Bagian berikut memberikan contoh pola filter untuk berbagai jenis sumber peristiwa. Mereka juga memberikan definisi format data masuk yang didukung dan format bodi pola filter untuk setiap layanan yang didukung.

- the section called “Pemfilteran peristiwa”
- the section called “Pemfilteran peristiwa”
- the section called “Pemfilteran peristiwa”
- the section called “Pemfilteran peristiwa”
- the section called “Pemfilteran peristiwa”

Menguji Fungsi Peristiwa Lambda di konsol

Anda dapat menguji Fungsi Peristiwa Lambda di konsol dengan menjalankan fungsi Anda dengan peristiwa pengujian. Peristiwa pengujian adalah masukan JSON ke Fungsi Acara Anda. Jika fungsi Anda tidak memerlukan input, acara dapat berupa dokumen kosong (`{}`).

Saat Anda menjalankan pengujian di konsol, Lambda secara sinkron memanggil fungsi Anda dengan peristiwa pengujian. Fungsi runtime mengubah acara JSON menjadi objek dan meneruskannya ke metode handler kode Anda untuk diproses.

> **Buat acara pengujian**
>
> Sebelum Anda dapat menguji di konsol, Anda perlu membuat acara pengujian pribadi atau yang dapat dibagikan.

Memanggil Fungsi Acara dengan acara pengujian

Untuk menguji Fungsi Peristiwa

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih nama Fungsi Acara yang ingin Anda uji.
3. Pilih tab Uji.
4. Di bawah Acara uji, pilih Buat acara baru atau Edit acara tersimpan, lalu pilih acara tersimpan yang ingin Anda gunakan.
5. Opsional - pilih Template untuk acara JSON.
6. Pilih Uji.
7. Untuk meninjau hasil pengujian, di bawah Hasil eksekusi, perluas Detail.

Untuk menjalankan Fungsi Acara Anda tanpa menyimpan acara pengujian Anda, pilih Uji sebelum menyimpan. Ini menciptakan peristiwa pengujian yang belum disimpan yang dipertahankan Lambda hanya selama sesi berlangsung.

Untuk runtime Node.js, Python, dan Ruby, Anda juga dapat mengakses peristiwa pengujian yang sudah disimpan dan belum disimpan di tab Kode. Gunakan bagian TEST EVENTS untuk membuat, mengedit, dan menjalankan tes.

Membuat acara pengujian pribadi

Acara pengujian pribadi hanya tersedia untuk pembuat acara, dan tidak memerlukan izin tambahan untuk digunakan. Anda dapat membuat dan menyimpan hingga 10 acara pengujian pribadi per Fungsi Acara.

Untuk membuat acara pengujian pribadi

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih nama Fungsi Acara yang ingin Anda uji.
3. Pilih tab Uji.
4. Di bawah acara Uji, lakukan hal berikut:
    a. Pilih Template.
    b. Masukkan Nama untuk tes.
    c. Di kotak entri teks, masukkan acara uji JSON.
    d. Di bawah Pengaturan berbagi acara, pilih Pribadi.
5. Pilih Simpan perubahan.

Untuk runtime Node.js, Python, dan Ruby, Anda juga dapat membuat peristiwa pengujian pada tab Kode. Gunakan bagian TEST EVENTS untuk membuat, mengedit, dan menjalankan tes.

Membuat acara uji yang dapat dibagikan

Peristiwa pengujian yang dapat dibagikan adalah peristiwa pengujian yang dapat Anda bagikan dengan pengguna lain di AWS akun yang sama. Anda dapat mengedit peristiwa pengujian yang dapat dibagikan pengguna lain dan memanggil Fungsi Acara Anda dengannya.

Lambda menyimpan peristiwa pengujian yang dapat dibagikan sebagai skema dalam registri skema Amazon EventBridge (CloudWatch Acara) bernama. `lambda-testevent-schemas` Karena Lambda menggunakan registri ini untuk menyimpan dan memanggil acara pengujian yang dapat dibagikan yang Anda buat, kami menyarankan Anda untuk tidak mengedit registri ini atau membuat registri menggunakan nama tersebut. `lambda-testevent-schemas`

Untuk melihat, membagikan, dan mengedit peristiwa pengujian yang dapat dibagikan, Anda harus memiliki izin untuk semua operasi API registri skema EventBridge (CloudWatch Peristiwa) berikut:

- skema. CreateRegistry

- skema. CreateSchema
- skema. DeleteSchema
- skema. DeleteSchemaVersion
- skema. DescribeRegistry
- skema. DescribeSchema
- skema. GetDiscoveredSchema
- skema. ListSchemaVersions
- skema. UpdateSchema

Harap diingat bahwa menyimpan pengeditan yang dilakukan pada peristiwa uji yang dapat dibagikan akan menimpa peristiwa tersebut.

Jika Anda tidak dapat membuat, mengedit, atau melihat peristiwa pengujian yang dapat dibagikan, periksa apakah akun Anda memiliki izin yang diperlukan untuk operasi ini. Jika Anda memiliki izin yang diperlukan tetapi masih tidak dapat mengakses peristiwa pengujian yang dapat dibagikan, periksa kebijakan berbasis sumber daya yang mungkin membatasi akses ke registri (Peristiwa). EventBridge CloudWatch

Untuk membuat acara pengujian yang dapat dibagikan

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih nama Fungsi Acara yang ingin Anda uji.
3. Pilih tab Uji.
4. Di bawah acara Uji, lakukan hal berikut:
    a. Pilih Template.
    b. Masukkan Nama untuk tes.
    c. Di kotak entri teks, masukkan acara uji JSON.
    d. Di bawah Pengaturan berbagi acara, pilih Dapat dibagikan.
5. Pilih Simpan perubahan.

Gunakan acara pengujian yang dapat dibagikan dengan AWS Serverless Application Model.

Anda dapat menggunakan AWS SAM untuk memanggil acara pengujian yang dapat dibagikan. Lihat acara uji jarak jauh sam di Panduan Pengembang AWS Serverless Application Model

Menghapus skema acara uji yang dapat dibagikan

Saat Anda menghapus peristiwa pengujian yang dapat dibagikan, Lambda menghapusnya dari `lambda-testevent-schemas` registri. Jika Anda menghapus peristiwa uji yang dapat dibagikan terakhir dari registri, Lambda akan menghapus registri tersebut.

Jika Anda menghapus Fungsi Peristiwa, Lambda tidak menghapus skema peristiwa pengujian yang dapat dibagikan terkait. Anda harus membersihkan sumber daya ini secara manual dari konsol EventBridge (CloudWatch Acara).

Status Fungsi Acara Lambda

Lambda menyertakan bidang Status dalam konfigurasi fungsi untuk semua Fungsi Peristiwa untuk menunjukkan kapan fungsi Anda siap untuk dipanggil. `State`memberikan informasi tentang status fungsi saat ini, termasuk apakah Anda berhasil menjalankan fungsi tersebut. Status Event Function tidak mengubah perilaku pemanggilan fungsi atau bagaimana fungsi Anda menjalankan kode.

> **Note**
>
> Definisi status Fungsi Acara sedikit berbeda untuk SnapStartfungsi. Untuk informasi selengkapnya, lihat Status Lambda SnapStart dan Fungsi Acara.

Dalam banyak kasus, tabel DynamoDB adalah cara ideal untuk mempertahankan status di antara pemanggilan karena menyediakan akses data latensi rendah dan dapat menskalakan dengan layanan Lambda. Anda juga dapat menyimpan data di Amazon EFS untuk Lambda jika Anda menggunakan layanan ini, dan ini menyediakan akses latensi rendah ke penyimpanan sistem file.

Status Fungsi Acara meliputi:

- `Pending`— Setelah Lambda membuat fungsi, ia menetapkan status ke pending. Saat dalam status tertunda, Lambda mencoba membuat atau mengonfigurasi sumber daya untuk fungsi tersebut, seperti sumber daya VPC atau EFS. Lambda tidak memanggil fungsi selama status tertunda. Setiap pemanggilan atau tindakan API lainnya yang beroperasi pada fungsi gagal.
- `Active`— Fungsi Anda bertransisi ke status aktif setelah Lambda menyelesaikan konfigurasi dan penyediaan sumber daya. Fungsi Acara hanya dapat berhasil dipanggil saat aktif.
- `Failed`— Menunjukkan bahwa konfigurasi atau penyediaan sumber daya mengalami kesalahan. Ketika pembuatan fungsi gagal, Lambda menyetel status fungsi menjadi gagal, dan Anda harus menghapus dan membuat ulang fungsi tersebut.
- `Inactive`— Sebuah fungsi menjadi tidak aktif ketika sudah menganggur cukup lama bagi Lambda untuk merebut kembali sumber daya eksternal yang dikonfigurasi untuknya. Saat Anda mencoba memanggil fungsi yang tidak aktif, pemanggilan gagal dan Lambda menyetel fungsi ke status tertunda hingga sumber daya fungsi dibuat ulang. Jika Lambda gagal membuat ulang sumber daya, fungsi kembali ke status tidak aktif. Anda mungkin perlu menyelesaikan kesalahan apa pun dan menerapkan kembali fungsi Anda untuk mengembalikannya ke status aktif.

  Fungsi juga transisi ke `Inactive` status jika Lambda kehilangan akses ke artefak sumber fungsi. Ini berlaku untuk fungsi gambar kontainer di mana gambar dihapus dari Amazon ECR atau izin

dicabut, dan ke fungsi.zip yang menggunakan penyimpanan kode S3 yang dikelola sendiri di mana objek sumber dihapus atau izin kebijakan bucket dicabut. Untuk mengembalikan fungsi, kembalikan akses ke artefak sumber dan perbarui fungsinya.

Jika Anda menggunakan alur kerja SDK-based otomatisasi atau memanggil API layanan Lambda secara langsung, pastikan Anda memeriksa status fungsi sebelum pemanggilan untuk memverifikasi bahwa fungsi tersebut aktif. Anda dapat melakukan ini dengan tindakan Lambda API GetFunction, atau dengan mengonfigurasi pelayan menggunakan SDK for AWS Java 2.0.

```
aws lambda get-function --function-name my-function --query 'Configuration.[State,
 LastUpdateStatus]'
```

Anda akan melihat output berikut:

```
[
 "Active",
 "Successful"
]
```

Operasi berikut gagal saat pembuatan fungsi tertunda:

- Memohon
- UpdateFunctionCode
- UpdateFunctionConfiguration
- PublishVersion

Status Fungsi Acara selama pembaruan

Lambda memiliki dua operasi untuk memperbarui Fungsi Acara:

- UpdateFunctionCode: Memperbarui paket penerapan fungsi
- UpdateFunctionConfiguration: Memperbarui konfigurasi fungsi

Lambda menggunakan LastUpdateStatusatribut untuk melacak kemajuan operasi pembaruan ini. Saat pembaruan sedang berlangsung (kapan`"LastUpdateStatus": "InProgress"`):

- Status fungsi tetap ada`Active`.

- Pemanggilan terus menggunakan kode dan konfigurasi fungsi sebelumnya hingga pembaruan selesai.
- Operasi berikut gagal:
  - UpdateFunctionCode
  - UpdateFunctionConfiguration
  - PublishVersion
  - TagResource

Saat pembaruan gagal (kapan`"LastUpdateStatus": "Failed"`):

- Status fungsi tetap ada`Active`.
- Pemanggilan terus menggunakan kode dan konfigurasi fungsi sebelumnya.

Example GetFunctionConfiguration respon

Contoh berikut adalah hasil GetFunctionConfigurationpermintaan pada fungsi yang menjalani pembaruan.

```
{
    "FunctionName": "my-function",
    "FunctionArn": "arn:aws:lambda:us-east-1:123456789012:function:my-function",
    "Runtime": "nodejs24.x",
    "VpcConfig": {
        "SubnetIds": [
            "subnet-071f712345678e7c8",
            "subnet-07fd123456788a036",
            "subnet-0804f77612345cacf"
        ],
        "SecurityGroupIds": [
            "sg-085912345678492fb"
        ],
        "VpcId": "vpc-08e1234569e011e83"
    },
    "State": "Active",
    "LastUpdateStatus": "InProgress",
    ...
}
```

Memahami perilaku coba lagi di Fungsi Peristiwa Lambda

Ketika Anda memanggil Fungsi Acara secara langsung, Anda menentukan strategi untuk menangani kesalahan yang terkait dengan kode fungsi Anda. Lambda tidak secara otomatis mencoba lagi jenis kesalahan ini atas nama Anda. Untuk mencoba lagi, Anda dapat memanggil ulang fungsi secara manual, mengirim peristiwa yang gagal ke antrian untuk debugging, atau mengabaikan kesalahan. Kode fungsi Anda mungkin telah berjalan sepenuhnya, sebagian, atau tidak sama sekali. Jika Anda mencoba lagi, pastikan bahwa kode fungsi Anda dapat menangani kejadian yang sama beberapa kali tanpa menyebabkan transaksi duplikat atau efek samping yang tidak diinginkan lainnya.

Saat Anda mengaktifkan suatu fungsi secara tidak langsung, Anda perlu menyadari perilaku dari invoker dan layanan apa pun yang dihadapi permintaan selama pelaksanaannya. Ini termasuk skenario berikut.

- Invokasi asinkron – Lambda mencoba kembali kesalahan fungsi dua kali. Jika fungsi tidak memiliki kapasitas yang cukup untuk menangani semua permintaan yang masuk, peristiwa mungkin menunggu dalam antrian selama berjam-jam untuk dikirim ke fungsi. Anda dapat mengonfigurasikan antrean dead-letter di fungsi untuk menangkap peristiwa yang tidak berhasil diproses. Untuk informasi selengkapnya, lihat the section called “Dead-letter antrian”.
- Pemetaan sumber peristiwa – Pemetaan sumber peristiwa yang membaca dari aliran mencoba kembali seluruh batch item. Kesalahan berulang memblokir pemrosesan shard terdampak hingga kesalahan diselesaikan atau item kedaluwarsa. Untuk mendeteksi shard yang terhambat, Anda dapat memantau metrik Usia Iterator.

  Untuk pemetaan sumber peristiwa yang membaca dari antrean, Anda menentukan lama waktu antara percobaan ulang dan destinasi untuk peristiwa yang gagal dengan mengonfigurasikan waktu habis tampilan dan kebijakan redrive pada antrean sumber. Untuk informasi selengkapnya, lihat Bagaimana Lambda memproses catatan dari sumber acara berbasis streaming dan antrian dan topik khusus layanan di bawah Memanggil Lambda dengan acara dari yang lain AWS layanan.
- AWS layanan — AWS layanan dapat memanggil fungsi Anda secara sinkron atau asinkron. Untuk invokasi sinkron, layanan memutuskan apakah akan mencoba kembali. Misalnya, operasi batch Amazon S3 mencoba ulang operasi jika Fungsi Peristiwa Lambda mengembalikan kode respons. `TemporaryFailure` Layanan yang meminta proxy dari pengguna atau klien hulu mungkin memiliki strategi coba lagi atau mungkin menyampaikan respons kesalahan kembali ke pemohon. Misalnya, API Gateway selalu menyampaikan respons kesalahan kembali ke pemohon.

  Untuk pemanggilan asinkron, logika coba lagi sama terlepas dari sumber pemanggilannya. Secara default, Lambda mencoba ulang pemanggilan asinkron yang gagal hingga dua kali. Untuk

informasi selengkapnya, lihat Bagaimana Lambda menangani kesalahan dan mencoba ulang dengan pemanggilan asinkron.

- Akun dan klien lain – Saat Anda memberikan akses ke akun lain, Anda dapat menggunakan kebijakan berbasis sumber daya untuk membatasi layanan atau sumber daya yang dapat mereka konfigurasikan untuk mengaktifkan fungsi Anda. Untuk melindungi fungsi Anda dari kelebihan beban, pertimbangkan untuk meletakkan lapisan API di depan fungsi Anda dengan Amazon API Gateway. Untuk informasi tentang penanganan error API Gateway, lihatthe section called "Kesalahan".

Untuk membantu Anda mengatasi kesalahan dalam aplikasi Lambda, Lambda terintegrasi dengan layanan seperti Amazon dan. CloudWatch AWS X-Ray Anda dapat menggunakan kombinasi log, metrik, alarm, dan pelacakan untuk mendeteksi dan mengidentifikasi masalah dengan cepat dalam kode fungsi Anda, API, atau sumber daya lain yang mendukung aplikasi Anda. Lihat informasi yang lebih lengkap di Pemantauan, debugging, dan pemecahan masalah Fungsi Peristiwa Lambda.

Menggunakan Deteksi Loop Rekursif Lambda untuk Mencegah Loop Tak Terbatas

Saat Anda mengonfigurasi Fungsi Peristiwa Lambda untuk menampilkan ke layanan atau sumber daya yang sama yang memanggil fungsi, dimungkinkan untuk membuat loop rekursif tak terbatas. Misalnya, Fungsi Peristiwa Lambda mungkin menulis pesan ke antrean Amazon Simple Queue Service (Amazon SQS), yang kemudian memanggil fungsi yang sama. Pemanggilan ini menyebabkan fungsi untuk menulis pesan lain ke antrian, yang pada gilirannya memanggil fungsi lagi.

Loop rekursif yang tidak disengaja dapat mengakibatkan biaya tak terduga ditagih ke Anda. Akun AWS Loop juga dapat menyebabkan Lambda menskalakan dan menggunakan semua konkurensi akun Anda yang tersedia. Untuk membantu mengurangi dampak loop yang tidak disengaja, Lambda mendeteksi jenis loop rekursif tertentu segera setelah terjadi.

Secara default, ketika Lambda mendeteksi loop rekursif, Lambda menghentikan fungsi Anda dipanggil dan memberi tahu Anda. Jika desain Anda sengaja menggunakan pola rekursif, Anda dapat mengubah konfigurasi default fungsi untuk memungkinkannya dipanggil secara rekursif. Untuk informasi selengkapnya, lihat the section called “Mengizinkan Fungsi Peristiwa Lambda berjalan dalam loop rekursif”.

Bagian-bagian

- Memahami deteksi loop rekursif
- Didukung Layanan AWS dan SDK
- Pemberitahuan loop rekursif
- Menanggapi notifikasi deteksi loop rekursif
- Mengizinkan Fungsi Peristiwa Lambda berjalan dalam loop rekursif
- Wilayah yang Didukung untuk deteksi loop rekursif Lambda

Memahami deteksi loop rekursif

Deteksi loop rekursif di Lambda bekerja dengan melacak peristiwa. Lambda adalah layanan komputasi berbasis peristiwa yang menjalankan kode fungsi Anda saat peristiwa tertentu terjadi. Misalnya, saat item ditambahkan ke topik antrian Amazon SQS atau Amazon Simple Notification Service (Amazon SNS). Lambda meneruskan peristiwa ke fungsi Anda sebagai objek JSON, yang

berisi informasi tentang perubahan status sistem. Ketika suatu peristiwa menyebabkan fungsi Anda berjalan, ini disebut pemanggilan.

Untuk mendeteksi loop rekursif, Lambda AWS X-Raymenggunakan header penelusuran. Ketika Layanan AWS mendukung deteksi loop rekursif mengirim peristiwa ke Lambda, peristiwa tersebut secara otomatis dianotasi dengan metadata. Saat Fungsi Peristiwa Lambda Anda menulis salah satu peristiwa ini ke peristiwa lain yang didukung Layanan AWS menggunakan versi AWS SDK yang didukung, fungsi ini akan memperbarui metadata ini. Metadata yang diperbarui mencakup hitungan berapa kali acara telah memanggil fungsi.

> **Note**
>
> Anda tidak perlu mengaktifkan penelusuran X-Ray aktif agar fitur ini berfungsi. Deteksi loop rekursif diaktifkan secara default untuk semua AWS pelanggan. Tidak ada biaya untuk menggunakan fitur ini.

Rantai permintaan adalah urutan pemanggilan Lambda yang disebabkan oleh peristiwa pemicu yang sama. Misalnya, bayangkan antrian Amazon SQS memanggil Fungsi Peristiwa Lambda Anda. Fungsi Peristiwa Lambda Anda kemudian mengirimkan peristiwa yang diproses kembali ke antrean Amazon SQS yang sama, yang memanggil fungsi Anda lagi. Dalam contoh ini, setiap pemanggilan fungsi Anda berada dalam rantai permintaan yang sama.

Jika fungsi Anda dipanggil sekitar 16 kali dalam rantai permintaan yang sama, maka Lambda secara otomatis menghentikan pemanggilan fungsi berikutnya dalam rantai permintaan tersebut dan memberi tahu Anda. Jika fungsi Anda dikonfigurasi dengan beberapa pemicu, maka pemanggilan dari pemicu lain tidak terpengaruh.

> **Note**
>
> Meskipun `maxReceiveCount` pengaturan pada kebijakan redrive antrian sumber lebih tinggi dari 16, perlindungan rekursi Lambda tidak mencegah Amazon SQS mencoba ulang pesan setelah loop rekursif terdeteksi dan dihentikan. Ketika Lambda mendeteksi loop rekursif dan menjatuhkan pemanggilan berikutnya, ia mengembalikan a `RecursiveInvocationException` ke pemetaan sumber peristiwa. Ini meningkatkan `receiveCount` nilai pada pesan. Lambda terus mencoba lagi pesan, dan terus memblokir pemanggilan fungsi, hingga Amazon SQS menentukan bahwa terlampaui dan mengirim pesan ke antrian `maxReceiveCount` huruf mati yang dikonfigurasi.

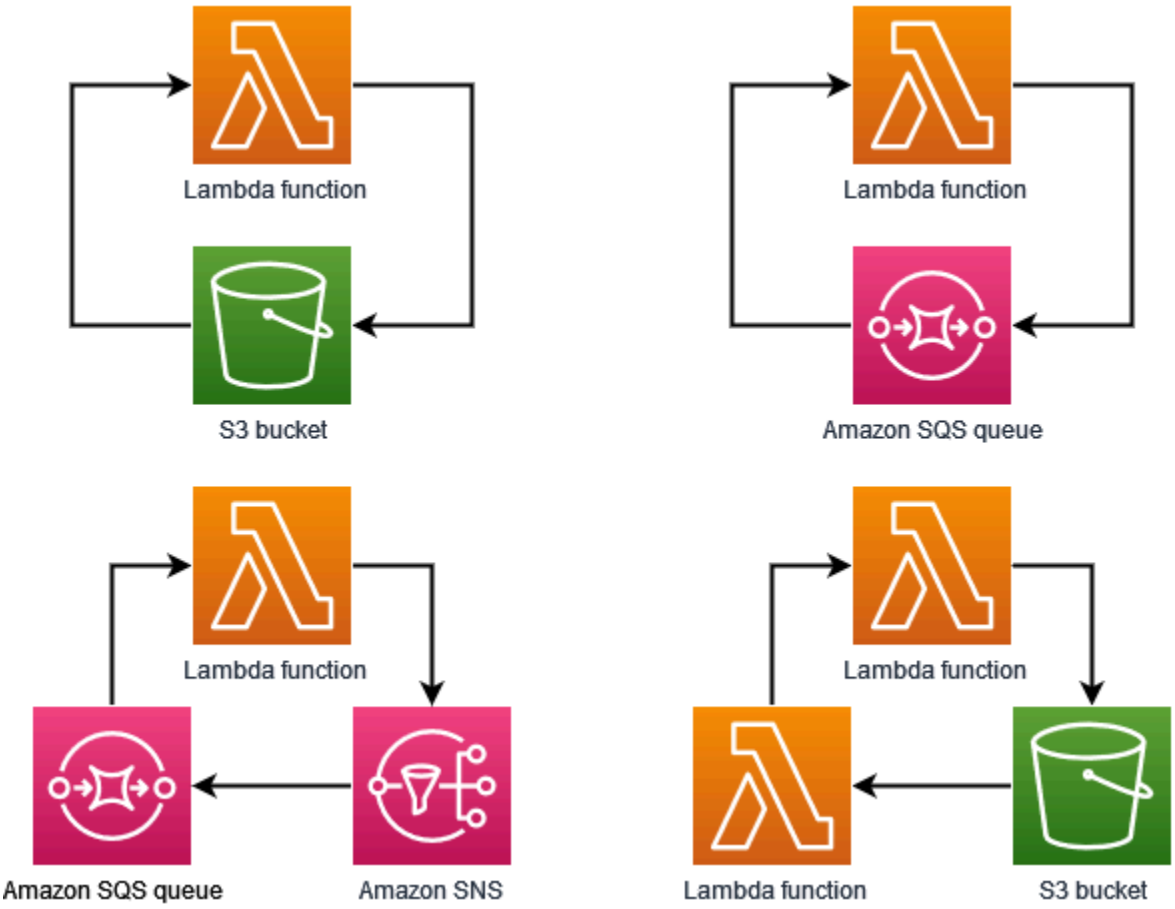
Jika Anda memiliki tujuan yang gagal atau antrean huruf mati yang dikonfigurasi untuk Fungsi Acara Anda, maka Lambda juga mengirimkan acara dari pemanggilan yang dihentikan ke antrean tujuan atau surat mati Anda. Saat mengonfigurasi antrian tujuan atau huruf mati untuk fungsi Anda, pastikan untuk tidak menggunakan pemicu peristiwa atau pemetaan sumber peristiwa yang juga digunakan fungsi Anda. Jika Anda mengirim peristiwa ke sumber daya yang sama yang memanggil fungsi Anda, maka Anda dapat membuat loop rekursif lain dan loop ini juga dihentikan. Jika Anda memilih keluar dari deteksi loop rekursi, loop ini tidak dihentikan.

Didukung Layanan AWS dan SDK

Lambda hanya dapat mendeteksi loop rekursif yang menyertakan dukungan tertentu. Layanan AWS Agar loop rekursif terdeteksi, Fungsi Acara Anda juga harus menggunakan salah satu AWS SDK yang didukung.

Didukung Layanan AWS

Lambda saat ini mendeteksi loop rekursif antara fungsi Anda, Amazon SQS, Amazon S3, Amazon SNS, dan bus acara Amazon Custom. EventBridge Lambda juga mendeteksi loop yang hanya terdiri dari Lambda Event Functions, yang mungkin memanggil satu sama lain secara sinkron atau asinkron. Diagram berikut menunjukkan beberapa contoh loop yang dapat dideteksi Lambda:



Ketika yang lain Layanan AWS seperti Amazon DynamoDB membentuk bagian dari loop, Lambda saat ini tidak dapat mendeteksi dan menghentikannya.

Karena Lambda saat ini hanya mendeteksi loop rekursif yang melibatkan Amazon SQS, Amazon S3, Amazon SNS, dan bus acara Kustom EventBridge Amazon, masih mungkin loop yang melibatkan Layanan AWS lainnya dapat mengakibatkan penggunaan Fungsi Peristiwa Lambda Anda yang tidak diinginkan.

Untuk mencegah tagihan tak terduga yang ditagih ke Anda Akun AWS, kami sarankan Anda mengonfigurasi CloudWatchalarm Amazon untuk mengingatkan Anda tentang pola penggunaan yang tidak biasa. Misalnya, Anda dapat mengonfigurasi CloudWatch untuk memberi tahu Anda tentang lonjakan dalam konkurensi atau pemanggilan Fungsi Peristiwa Lambda. Anda juga dapat mengonfigurasi alarm penagihan untuk memberi tahu Anda saat pengeluaran di akun melebihi ambang batas yang Anda tentukan. Atau, Anda dapat menggunakannya AWS Cost Anomaly Detectionuntuk mengingatkan Anda tentang pola penagihan yang tidak biasa.

Didukung AWS SDK

Agar Lambda dapat mendeteksi loop rekursif, fungsi Anda harus menggunakan salah satu versi SDK berikut atau yang lebih tinggi:

Waktu Aktif	Versi AWS SDK minimum yang diperlukan
Node.js	2.1147.0 (SDK versi 2) 3.105.0 (SDK versi 3)
Python	1.24.46 (Boto3) 1.27.46 (botocore)
Java 8 dan Java 11	2.17.135
Jawa 17	2.20.81
Jawa 21	2.21.24
.NET	3.7.293.0
Ruby	3.134.0
PHP	3.232.0
Go	V2 SDK 1,57,0

Beberapa runtime Lambda seperti Python dan Node.js menyertakan versi SDK. AWS Jika versi SDK yang disertakan dalam runtime fungsi Anda lebih rendah dari minimum yang diperlukan, Anda dapat menambahkan versi SDK yang didukung ke paket penerapan fungsi Anda. Anda juga dapat menambahkan versi SDK yang didukung ke fungsi Anda menggunakan lapisan Lambda. Untuk daftar SDK yang disertakan dengan setiap runtime Lambda, lihat. Runtime Fungsi Peristiwa Lambda

Pemberitahuan loop rekursif

Ketika Lambda menghentikan loop rekursif, Anda menerima pemberitahuan melalui Dasbor Healthdan melalui email. Anda juga dapat menggunakan CloudWatch metrik untuk memantau jumlah pemanggilan rekursif yang dihentikan Lambda.

Dasbor Health notifikasi

Ketika Lambda menghentikan pemanggilan rekursif, akan Dasbor Health menampilkan pemberitahuan di halaman kesehatan akun Anda, di bawah Terbuka dan masalah terbaru. Perhatikan bahwa ini dapat memakan waktu hingga 3,5 jam setelah Lambda menghentikan pemanggilan rekursif sebelum pemberitahuan ini ditampilkan. Untuk informasi selengkapnya tentang melihat peristiwa akun di Dasbor Health, lihat Memulai dengan Dasbor AWS Kesehatan — Kesehatan akun Anda di Panduan Pengguna AWS Kesehatan.

Peringatan email

Saat Lambda pertama kali menghentikan pemanggilan rekursif Fungsi Acara Anda, Lambda mengirimkan peringatan email kepada Anda. Lambda mengirim maksimal satu email setiap 24 jam untuk setiap fungsi di Anda. Akun AWS Setelah Lambda mengirimkan pemberitahuan email, Anda tidak akan menerima email lagi untuk fungsi itu selama 24 jam lagi, bahkan jika Lambda menghentikan pemanggilan rekursif lebih lanjut dari fungsi tersebut. Perhatikan bahwa ini dapat memakan waktu hingga 3,5 jam setelah Lambda menghentikan pemanggilan rekursif sebelum Anda menerima peringatan email ini.

Lambda mengirimkan peringatan email loop rekursif ke kontak akun utama Anda Akun AWS dan kontak operasi alternatif Anda. Untuk informasi tentang melihat atau memperbarui alamat email di akun Anda, lihat Memperbarui informasi kontak di Referensi AWS Umum.

CloudWatch Metrik Amazon

CloudWatch Metrik `RecursiveInvocationsDropped` mencatat jumlah pemanggilan fungsi yang dihentikan Lambda karena fungsi Anda telah dipanggil lebih dari sekitar 16 kali dalam satu rantai permintaan. Lambda memancarkan metrik ini segera setelah menghentikan pemanggilan rekursif. Untuk melihat metrik ini, ikuti petunjuk untuk Melihat metrik di CloudWatch konsol dan pilih metrik`RecursiveInvocationsDropped`.

Menanggapi notifikasi deteksi loop rekursif

Ketika Fungsi Peristiwa Anda dipanggil lebih dari sekitar 16 kali oleh peristiwa pemicu yang sama, Lambda menghentikan pemanggilan fungsi berikutnya untuk acara tersebut untuk memutus loop rekursif. Untuk mencegah terulangnya loop rekursif yang rusak Lambda, lakukan hal berikut:

- Kurangi konkurensi fungsi Anda yang tersedia menjadi nol, yang membatasi semua pemanggilan future.

- Hapus atau nonaktifkan pemicu atau pemetaan sumber peristiwa yang menjalankan fungsi Anda.
- Identifikasi dan perbaiki cacat kode yang menulis peristiwa kembali ke AWS sumber daya yang menjalankan fungsi Anda. Sebuah sumber umum cacat terjadi ketika Anda menggunakan variabel untuk menentukan sumber peristiwa fungsi dan target. Periksa apakah Anda tidak menggunakan nilai yang sama untuk kedua variabel.

Selain itu, jika sumber peristiwa untuk Fungsi Peristiwa Lambda Anda adalah antrean Amazon SQS, pertimbangkan untuk mengonfigurasi antrian huruf mati pada antrean sumber.

> **Note**
>
> Pastikan Anda mengonfigurasi antrian huruf mati pada antrean sumber, bukan pada Fungsi Peristiwa Lambda. Antrean surat gagal yang dikonfigurasikan di fungsi digunakan untuk antrean invokasi asinkron fungsi, bukan untuk antrean sumber kejadian.

Jika sumber acara adalah topik Amazon SNS, pertimbangkan untuk menambahkan tujuan yang gagal untuk fungsi Anda.

Untuk mengurangi konkurensi Fungsi Acara yang tersedia menjadi nol (konsol)

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih nama Fungsi Acara Anda.
3. Pilih Throttle.
4. Di kotak dialog Throttle your function, pilih Confirm.

Untuk menghapus pemicu atau pemetaan sumber peristiwa untuk Fungsi Acara (konsol)

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih nama Fungsi Acara Anda.
3. Pilih tab Konfigurasi, lalu pilih Pemicu.
4. Di bawah Pemicu, pilih pemicu atau pemetaan sumber peristiwa yang ingin Anda hapus, lalu pilih Hapus.
5. Di kotak dialog Hapus pemicu, pilih Hapus.

Untuk menonaktifkan pemetaan sumber peristiwa untuk Fungsi Acara Anda (AWS CLI)

1. Untuk menemukan UUID untuk pemetaan sumber peristiwa yang ingin Anda nonaktifkan, jalankan perintah AWS Command Line Interface (AWS CLI) list-event-source-mappings.

```
aws lambda list-event-source-mappings
```

2. Untuk menonaktifkan pemetaan sumber peristiwa, jalankan perintah AWS CLI update-event-source-mapping berikut.

```
aws lambda update-event-source-mapping --function-name MyFunction \
--uuid a1b2c3d4-5678-90ab-cdef-EXAMPLE11111 --no-enabled
```

Mengizinkan Fungsi Peristiwa Lambda berjalan dalam loop rekursif

Jika desain Anda sengaja menggunakan loop rekursif, Anda dapat mengonfigurasi Fungsi Peristiwa Lambda untuk memungkinkannya dipanggil secara rekursif. Kami menyarankan Anda menghindari penggunaan loop rekursif dalam desain Anda. Kesalahan implementasi dapat menyebabkan pemanggilan rekursif menggunakan semua konkurensi Anda Akun AWS yang tersedia dan biaya tak terduga yang ditagih ke akun Anda.

> **Important**
>
> Jika Anda menggunakan loop rekursif, perlakukan dengan hati-hati. Menerapkan best practice guard rail untuk meminimalkan risiko kesalahan implementasi. Untuk mempelajari lebih lanjut tentang praktik terbaik untuk menggunakan pola rekursif, lihat Pola rekursif yang menyebabkan fungsi Lambda yang tidak terkendali di Tanah Tanpa Server.

Anda dapat mengonfigurasi fungsi untuk mengizinkan loop rekursif menggunakan konsol Lambda, AWS Command Line Interface AWS CLI(), dan PutFunctionRecursionConfigAPI. Anda juga dapat mengonfigurasi pengaturan deteksi loop rekursif fungsi di AWS SAM dan CloudFormation.

Secara default, Lambda mendeteksi dan mengakhiri loop rekursif. Kecuali desain Anda sengaja menggunakan loop rekursif, sebaiknya Anda tidak mengubah konfigurasi default fungsi Anda.

Perhatikan bahwa saat Anda mengonfigurasi fungsi untuk mengizinkan loop rekursif, CloudWatch metrik `RecursiveInvocationsDropped` tidak dipancarkan.

Console

Untuk memungkinkan Fungsi Peristiwa berjalan dalam loop rekursif (konsol)

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih nama Fungsi Acara Anda untuk membuka halaman detail fungsi.
3. Pilih tab Konfigurasi, lalu pilih Konkurensi dan deteksi rekursi.
4. Selain deteksi loop rekursif, pilih Edit.
5. Pilih Izinkan loop rekursif.
6. Pilih Simpan.

AWS CLI

Anda dapat menggunakan PutFunctionRecursionConfigAPI untuk memungkinkan fungsi Anda dipanggil dalam loop rekursif. Tentukan `Allow` untuk parameter loop rekursif. Misalnya, Anda dapat memanggil API ini dengan `put-function-recursion-config` AWS CLI perintah:

```
aws lambda put-function-recursion-config --function-name yourFunctionName --
recursive-loop Allow
```

Anda dapat mengubah konfigurasi Fungsi Acara kembali ke pengaturan default sehingga Lambda menghentikan loop rekursif saat mendeteksi mereka. Edit konfigurasi fungsi Anda menggunakan konsol Lambda atau. AWS CLI

Console

Untuk mengkonfigurasi Fungsi Peristiwa sehingga loop rekursif dihentikan (konsol)

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih nama Fungsi Acara Anda untuk membuka halaman detail fungsi.
3. Pilih tab Konfigurasi, lalu pilih Konkurensi dan deteksi rekursi.
4. Selain deteksi loop rekursif, pilih Edit.
5. Pilih Hentikan loop rekursif.
6. Pilih Simpan.

AWS CLI

Anda dapat menggunakan PutFunctionRecursionConfigAPI untuk mengonfigurasi fungsi Anda sehingga Lambda menghentikan loop rekursif saat mendeteksi mereka. Tentukan `Terminate` untuk parameter loop rekursif. Misalnya, Anda dapat memanggil API ini dengan `put-function-recursion-config` AWS CLI perintah:

```
aws lambda put-function-recursion-config --function-name yourFunctionName --
recursive-loop Terminate
```

Wilayah yang Didukung untuk deteksi loop rekursif Lambda

Deteksi loop rekursif Lambda didukung di semua Wilayah komersial.

Membuat dan mengelola URL fungsi Lambda

URL fungsi adalah titik akhir HTTP (S) khusus untuk Fungsi Acara Lambda Anda. Anda dapat membuat dan mengonfigurasi URL fungsi melalui konsol Lambda atau API Lambda.

> **Tip**
>
> Lambda menawarkan dua cara untuk menjalankan fungsi Anda melalui titik akhir HTTP: URL fungsi dan Amazon API Gateway. Jika Anda tidak yakin metode mana yang terbaik untuk kasus penggunaan Anda, lihatthe section called “URL Fungsi vs Amazon API Gateway”.

Saat Anda membuat URL fungsi, Lambda secara otomatis menghasilkan titik akhir URL unik untuk Anda. Setelah Anda membuat URL fungsi, titik akhir URL-nya tidak pernah berubah. Fungsi titik akhir URL memiliki format berikut:

```
https://<url-id>.lambda-url.<region>.on.aws
```

> **Note**
>
> URL fungsi tidak didukung di semua AWS wilayah. Untuk memeriksa ketersediaan, lihat Layanan AWS Regional dan filter berdasarkan`Feature Name = Function URLs`.

URL fungsi adalah dual stack-enabled, mendukung IPv4 dan IPv6. Setelah Anda mengkonfigurasi URL fungsi untuk fungsi Anda, Anda dapat memanggil fungsi Anda melalui titik akhir HTTP (S) dengan menggunakan browser web, curl, Postman, atau klien HTTP apa pun.

> **Note**
>
> Anda dapat mengakses URL fungsi Anda hanya melalui Internet publik. Sementara Lambda Event Functions mendukung AWS PrivateLink, URL fungsi tidak.

URL fungsi Lambda menggunakan kebijakan berbasis sumber daya untuk keamanan dan kontrol akses. URL fungsi juga mendukung opsi konfigurasi cross-origin resource sharing (CORS).

Anda dapat menerapkan URL fungsi ke alias fungsi apa pun, atau ke versi fungsi yang `$LATEST` tidak dipublikasikan. Anda tidak dapat menambahkan URL fungsi ke versi fungsi lainnya.

Bagian berikut menunjukkan cara membuat dan mengelola URL fungsi menggunakan konsol Lambda, AWS CLI, dan CloudFormation template. Untuk tutorial langkah demi langkah tentang penggunaan URL fungsi sebagai titik akhir webhook, lihat. the section called “Tutorial: Membuat titik akhir webhook”

Topik

- Membuat URL fungsi (konsol)
- Membuat URL fungsi (AWS CLI)
- Menambahkan URL fungsi ke CloudFormation template
- Cross-origin berbagi sumber daya (CORS)
- URL fungsi pelambatan
- Menonaktifkan URL fungsi
- Menghapus URL fungsi
- Kontrol akses ke URL fungsi Lambda
- Memanggil URL fungsi Lambda
- Memantau fungsi Lambda URLs
- Pilih metode untuk memanggil Fungsi Peristiwa Lambda Anda menggunakan permintaan HTTP
- Tutorial: Membuat titik akhir webhook menggunakan URL fungsi Lambda

Membuat URL fungsi (konsol)

Ikuti langkah-langkah ini untuk membuat URL fungsi menggunakan konsol.

Untuk membuat URL fungsi untuk fungsi yang ada

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih nama Fungsi Acara yang ingin Anda buat URL fungsi.
3. Pilih tab Konfigurasi, lalu pilih URL Fungsi.
4. Pilih Buat URL fungsi.
5. Untuk jenis Auth, pilih AWS_IAM atau NONE. Untuk informasi selengkapnya tentang otentikasi URL fungsi, lihatKontrol akses.
6. (Opsional) Pilih Konfigurasi berbagi sumber daya lintas asal (CORS), lalu konfigurasikan pengaturan CORS untuk URL fungsi Anda. Untuk informasi selengkapnya tentang CORS, lihat Cross-origin berbagi sumber daya (CORS).

7. Pilih Simpan.

Ini membuat URL fungsi untuk versi fungsi Anda `$LATEST` yang tidak dipublikasikan. URL fungsi muncul di bagian Ikhtisar fungsi konsol.

Untuk membuat URL fungsi untuk alias yang ada

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih nama Fungsi Acara dengan alias yang ingin Anda buat URL fungsi.
3. Pilih tab Alias, lalu pilih nama alias yang ingin Anda buat URL fungsinya.
4. Pilih tab Konfigurasi, lalu pilih URL Fungsi.
5. Pilih Buat URL fungsi.
6. Untuk jenis Auth, pilih AWS_IAM atau NONE. Untuk informasi selengkapnya tentang otentikasi URL fungsi, lihatKontrol akses.
7. (Opsional) Pilih Konfigurasi berbagi sumber daya lintas asal (CORS), lalu konfigurasikan pengaturan CORS untuk URL fungsi Anda. Untuk informasi selengkapnya tentang CORS, lihat Cross-origin berbagi sumber daya (CORS).
8. Pilih Simpan.

Ini membuat URL fungsi untuk alias fungsi Anda. URL fungsi muncul di bagian Ikhtisar fungsi konsol untuk alias Anda.

Untuk membuat fungsi baru dengan URL fungsi

Untuk membuat Fungsi Acara baru dengan URL fungsi (konsol)

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih Buat Fungsi Acara.
3. Di bagian Informasi dasar, lakukan hal berikut:
    a. Untuk nama Fungsi, masukkan nama untuk fungsi Anda, seperti**`my-function`**.
    b. Untuk Runtime, pilih runtime bahasa yang Anda inginkan, seperti Node.js 24.
    c. Untuk Arsitektur, pilih x86_64 atau arm64.
    d. Perluas Izin, lalu pilih apakah akan membuat peran eksekusi baru atau menggunakan yang sudah ada.

4. Perluas Pengaturan lanjutan, lalu pilih URL Fungsi.
5. Untuk jenis Auth, pilih AWS_IAM atau NONE. Untuk informasi selengkapnya tentang otentikasi URL fungsi, lihatKontrol akses.
6. (Opsional) Pilih Konfigurasi berbagi sumber daya lintas asal (CORS). Dengan memilih opsi ini selama pembuatan fungsi, URL fungsi Anda memungkinkan permintaan dari semua asal secara default. Anda dapat mengedit pengaturan CORS untuk URL fungsi Anda setelah membuat fungsi. Untuk informasi selengkapnya tentang CORS, lihat Cross-origin berbagi sumber daya (CORS).
7. Pilih Buat Fungsi Acara.

Ini menciptakan fungsi baru dengan URL fungsi untuk versi fungsi `$LATEST` yang tidak dipublikasikan. URL fungsi muncul di bagian Ikhtisar fungsi konsol.

Membuat URL fungsi (AWS CLI)

Untuk membuat URL fungsi untuk Fungsi Peristiwa Lambda yang ada menggunakan AWS Command Line Interface (AWS CLI), jalankan perintah berikut:

```
aws lambda create-function-url-config \
    --function-name my-function \
    --qualifier prod \ // optional
    --auth-type AWS_IAM
    --cors-config {AllowOrigins="https://example.com"} // optional
```

Ini menambahkan URL fungsi ke **`prod`** qualifier untuk fungsi **`my-function`** tersebut. Untuk informasi selengkapnya tentang parameter konfigurasi ini, lihat CreateFunctionUrlConfigdi referensi API.

> **Note**
>
> Untuk membuat URL fungsi dengan menggunakan AWS CLI, fungsi tersebut harus sudah ada.

Menambahkan URL fungsi ke CloudFormation template

Untuk menambahkan `AWS::Lambda::Url` sumber daya ke CloudFormation template Anda, gunakan sintaks berikut:

JSON

```
{
  "Type" : "AWS::Lambda::Url",
  "Properties" : {
      "AuthType" : String,
      "Cors" : Cors,
      "Qualifier" : String,
      "TargetFunctionArn" : String
    }
}
```

YAML

```
Type: AWS::Lambda::Url
Properties:
  AuthType: String
  Cors:
    Cors
  Qualifier: String
  TargetFunctionArn: String
```

Parameter

- (Wajib) `AuthType` — Mendefinisikan jenis otentikasi untuk URL fungsi Anda. Nilai yang mungkin adalah salah satu `AWS_IAM` atau`NONE`. Untuk membatasi akses ke pengguna yang diautentikasi saja, setel ke. `AWS_IAM` Untuk melewati otentikasi IAM dan memungkinkan pengguna untuk membuat permintaan ke fungsi Anda, atur ke. `NONE`
- (Opsional) `Cors` - Mendefinisikan pengaturan CORS untuk URL fungsi Anda. `Cors`Untuk menambah `AWS::Lambda::Url` sumber daya Anda CloudFormation, gunakan sintaks berikut.

  Example AWS::Lambda:: Url.Cors (JSON)

  ```
  {
    "AllowCredentials" : Boolean,
    "AllowHeaders" : [ String, ... ],
    "AllowMethods" : [ String, ... ],
    "AllowOrigins" : [ String, ... ],
    "ExposeHeaders" : [ String, ... ],
    "MaxAge" : Integer
  ```

```
}
```

Example AWS::Lambda:: Url.Cors (YAMAL)

```
AllowCredentials: Boolean
AllowHeaders:
  - String
AllowMethods:
  - String
AllowOrigins:
  - String
ExposeHeaders:
  - String
MaxAge: Integer
```

- (Opsional) `Qualifier` — Nama alias.
- (Wajib) `TargetFunctionArn` - Nama atau Nama Sumber Daya Amazon (ARN) dari Fungsi Peristiwa Lambda. Format nama yang valid meliputi:
  - Nama fungsi - `my-function`
  - Fungsi ARN — `arn:aws:lambda:us-west-2:123456789012:function:my-function`
  - ARN Sebagian — `123456789012:function:my-function`

Cross-origin berbagi sumber daya (CORS)

Untuk menentukan bagaimana asal yang berbeda dapat mengakses URL fungsi Anda, gunakan berbagi sumber daya lintas asal (CORS). Sebaiknya konfigurasi CORS jika Anda bermaksud memanggil URL fungsi Anda dari domain yang berbeda. Lambda mendukung header CORS berikut untuk URL fungsi.

Sundulan CORS	Properti konfigurasi CORS	Contoh nilai

Access-Control-Allow-Origin	`AllowOrigins`	`*`(izinkan semua asal)
		`https://www.example.com`
		`http://localhost:60905`

Sundulan CORS	Properti konfigurasi CORS	Contoh nilai
Access-Control-Allow-Methods	`AllowMethods`	`GET`, `POST`, `DELETE`, `*`
Access-Control-Allow-Headers	`AllowHeaders`	`Date`, `Keep-Alive` , `X-Custom-Header`
Access-Control-Expose-Headers	`ExposeHeaders`	`Date`, `Keep-Alive` , `X-Custom-Header`
Access-Control-Allow-Credentials	`AllowCredentials`	`TRUE`
Access-Control-Max-Age	`MaxAge`	`5` (default), `300`

Saat Anda mengonfigurasi CORS untuk URL fungsi menggunakan konsol Lambda atau konsol AWS CLI, Lambda secara otomatis menambahkan header CORS ke semua respons melalui URL fungsi. Atau, Anda dapat menambahkan header CORS secara manual ke respons fungsi Anda. Jika ada header yang bertentangan, perilaku yang diharapkan tergantung pada jenis permintaan:

- Untuk permintaan preflight seperti permintaan OPTIONS, header CORS yang dikonfigurasi pada URL fungsi diutamakan. Lambda hanya mengembalikan header CORS ini dalam respons.
- Untuk permintaan non-preflight seperti permintaan GET atau POST, Lambda mengembalikan header CORS yang dikonfigurasi pada URL fungsi, serta header CORS yang dikembalikan oleh fungsi. Ini dapat menghasilkan duplikat header CORS dalam respons. Anda mungkin melihat kesalahan yang mirip dengan berikut ini:`The 'Access-Control-Allow-Origin' header contains multiple values '*, *', but only one is allowed.`

Secara umum, kami menyarankan untuk mengonfigurasi semua pengaturan CORS pada URL fungsi, daripada mengirim header CORS secara manual dalam respons fungsi.

URL fungsi pelambatan

Throttling membatasi tingkat permintaan proses fungsi Anda. Ini berguna dalam banyak situasi, seperti mencegah fungsi Anda membebani sumber daya hilir, atau menangani lonjakan permintaan yang tiba-tiba.

Anda dapat membatasi laju permintaan yang diproses oleh Fungsi Peristiwa Lambda Anda melalui URL fungsi dengan mengonfigurasi konkurensi cadangan. Konkurensi cadangan membatasi jumlah pemanggilan bersamaan maksimum untuk fungsi Anda. Tingkat permintaan maksimum per detik (RPS) fungsi Anda setara dengan 10 kali konkurensi cadangan yang dikonfigurasi. Misalnya, jika Anda mengonfigurasi fungsi Anda dengan konkurensi cadangan 100, maka RPS maksimum adalah 1.000.

Setiap kali konkurensi fungsi Anda melebihi konkurensi cadangan, URL fungsi Anda mengembalikan kode 429 status HTTP. Jika fungsi Anda menerima permintaan yang melebihi maksimum 10x RPS berdasarkan konkurensi cadangan yang dikonfigurasi, Anda juga menerima kesalahan HTTP. 429 Untuk informasi selengkapnya tentang konkurensi cadangan, lihatMengkonfigurasi konkurensi cadangan untuk Fungsi Acara.

Menonaktifkan URL fungsi

Dalam keadaan darurat, Anda mungkin ingin menolak semua lalu lintas ke URL fungsi Anda. Untuk menonaktifkan URL fungsi Anda, atur konkurensi cadangan ke nol. Ini membatasi semua permintaan ke URL fungsi Anda, menghasilkan respons 429 status HTTP. Untuk mengaktifkan kembali URL fungsi Anda, hapus konfigurasi konkurensi cadangan, atau atur konfigurasi ke jumlah yang lebih besar dari nol.

Menghapus URL fungsi

Saat Anda menghapus URL fungsi, Anda tidak dapat memulihkannya. Membuat URL fungsi baru menghasilkan alamat URL yang berbeda.

> **Note**
>
> Jika Anda menghapus URL fungsi dengan jenis autentikasi`NONE`, Lambda tidak secara otomatis menghapus kebijakan berbasis sumber daya terkait. Jika ingin menghapus kebijakan ini, Anda harus melakukannya secara manual.

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih nama Fungsi Acara.
3. Pilih tab Konfigurasi, lalu pilih URL Fungsi.
4. Pilih Hapus.

5. Masukkan kata hapus ke dalam bidang untuk mengonfirmasi penghapusan.
6. Pilih Hapus.

> **Note**
>
> Saat Anda menghapus fungsi yang memiliki URL fungsi, Lambda menghapus URL fungsi secara asinkron. Jika Anda segera membuat fungsi baru dengan nama yang sama di akun yang sama, ada kemungkinan URL fungsi asli dipetakan ke fungsi baru alih-alih dihapus.

Kontrol akses ke URL fungsi Lambda

> **Note**
>
> Mulai Oktober 2025, URL fungsi baru akan memerlukan keduanya `lambda:InvokeFunctionUrl` dan `lambda:InvokeFunction` izin.

Anda dapat mengontrol akses ke URL fungsi Lambda menggunakan AuthTypeparameter yang dikombinasikan dengan kebijakan berbasis sumber daya yang dilampirkan ke Fungsi Peristiwa spesifik Anda. Konfigurasi kedua komponen ini menentukan siapa yang dapat memanggil atau melakukan tindakan administratif lainnya pada URL fungsi Anda.

`AuthType`Parameter menentukan cara Lambda mengautentikasi atau mengotorisasi permintaan ke URL fungsi Anda. Saat Anda mengonfigurasi URL fungsi Anda, Anda harus menentukan salah satu `AuthType` opsi berikut:

- `AWS_IAM`— Lambda menggunakan AWS Identity and Access Management (IAM) untuk mengautentikasi dan mengotorisasi permintaan berdasarkan kebijakan identitas kepala sekolah IAM dan kebijakan berbasis sumber daya fungsi. Pilih opsi ini jika Anda hanya ingin pengguna dan peran yang diautentikasi untuk memanggil fungsi Anda menggunakan URL fungsi.

- `NONE`— Lambda tidak melakukan otentikasi apa pun sebelum menjalankan fungsi Anda. Namun, kebijakan berbasis sumber daya fungsi Anda selalu berlaku dan harus memberikan akses publik sebelum URL fungsi Anda dapat menerima permintaan. Pilih opsi ini untuk mengizinkan akses publik yang tidak diautentikasi ke URL fungsi Anda.

Untuk wawasan tambahan tentang keamanan, Anda dapat menggunakan AWS Identity and Access Management Access Analyzer untuk mendapatkan analisis komprehensif tentang akses eksternal ke URL fungsi Anda. IAM Access Analyzer juga memantau izin baru atau yang diperbarui pada Fungsi Acara Lambda Anda untuk membantu Anda mengidentifikasi izin yang memberikan akses publik dan lintas akun. Anda dapat menggunakan IAM Access Analyzer tanpa biaya. Untuk memulai dengan IAM Access Analyzer, lihat Menggunakan AWS IAM Access Analyzer.

Halaman ini berisi contoh kebijakan berbasis sumber daya untuk kedua jenis autentikasi, dan cara membuat kebijakan ini menggunakan AddPermissionoperasi API atau konsol Lambda. Untuk informasi tentang cara memanggil URL fungsi setelah menyiapkan izin, lihat. Memanggil URL fungsi Lambda

Topik

- Menggunakan tipe autentikasi AWS_IAM
- Menggunakan tipe autentikasi NONE
- Tata kelola dan kontrol akses

Menggunakan tipe autentikasi **`AWS_IAM`**

Jika Anda memilih jenis `AWS_IAM` autentikasi, pengguna yang perlu memanggil URL fungsi Lambda Anda harus memiliki izin dan. `lambda:InvokeFunctionUrl` `lambda:InvokeFunction` Bergantung pada siapa yang membuat permintaan pemanggilan, Anda mungkin harus memberikan izin ini menggunakan kebijakan berbasis sumber daya.

Jika prinsipal yang membuat permintaan Akun AWS sama dengan URL fungsi, maka prinsipal harus memiliki **`lambda:InvokeFunctionUrl`** dan **`lambda:InvokeFunction`** izin dalam kebijakan berbasis identitas mereka, atau memiliki izin yang diberikan kepada mereka dalam kebijakan berbasis sumber daya fungsi. Dengan kata lain, kebijakan berbasis sumber daya bersifat opsional jika pengguna sudah memiliki `lambda:InvokeFunctionUrl` dan `lambda:InvokeFunction` izin dalam kebijakan berbasis identitas mereka. Evaluasi kebijakan mengikuti aturan yang diuraikan dalam logika evaluasi kebijakan.

Jika prinsipal yang membuat permintaan berada di akun yang berbeda, maka prinsipal harus memiliki kebijakan berbasis identitas yang memberi mereka **`lambda:InvokeFunctionUrl`** dan izin serta **`lambda:InvokeFunction`** izin yang diberikan kepada mereka dalam kebijakan berbasis sumber daya pada fungsi yang mereka coba panggil. Evaluasi kebijakan mengikuti aturan yang diuraikan dalam Menentukan apakah permintaan lintas akun diperbolehkan.

Kebijakan berbasis sumber daya berikut memungkinkan `example` peran Akun AWS `444455556666` untuk memanggil URL fungsi yang terkait dengan fungsi. `my-function` Kunci InvokedViaFunctionUrl konteks lambda: membatasi `lambda:InvokeFunction` tindakan untuk berfungsi panggilan URL. Ini berarti bahwa prinsipal harus menggunakan URL fungsi untuk memanggil fungsi. Jika Anda tidak menyertakan`lambda:InvokedViaFunctionUrl`, prinsipal dapat memanggil fungsi Anda melalui metode pemanggilan lainnya, selain URL fungsi.

Example— kebijakan Cross-account berbasis sumber daya

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::444455556666:role/example"
      },
      "Action": "lambda:InvokeFunctionUrl",
      "Resource": "arn:aws:lambda:us-east-1:123456789012:function:my-function",
      "Condition": {
        "StringEquals": {
          "lambda:FunctionUrlAuthType": "AWS_IAM"
        }
      }
    },
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::444455556666:role/example"
      },
      "Action": "lambda:InvokeFunction",
      "Resource": "arn:aws:lambda:us-east-1:123456789012:function:my-function",
      "Condition": {
        "Bool": {
          "lambda:InvokedViaFunctionUrl": "true"
        }
      }
    }
  ]
}
```

Anda dapat membuat kebijakan berbasis sumber daya ini melalui konsol menggunakan langkah-langkah berikut:

Untuk memberikan izin pemanggilan URL ke akun lain (konsol)

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih nama fungsi yang ingin Anda berikan izin pemanggilan URL.
3. Pilih tab Konfigurasi, lalu pilih Izin.
4. Di bawah Resource-based kebijakan, pilih Tambahkan izin.
5. Pilih URL Fungsi.
6. Untuk jenis Auth, pilih AWS_IAM.
7. Masukkan ID Pernyataan untuk pernyataan kebijakan Anda.
8. Untuk Principal, masukkan ID akun atau Nama Sumber Daya Amazon (ARN) pengguna atau peran yang ingin Anda berikan izin. Sebagai contoh: **444455556666**.
9. Pilih Simpan.

Atau, Anda dapat membuat kebijakan ini menggunakan perintah add-permission AWS Command Line Interface (AWS CLI) berikut. Saat Anda menggunakan AWS CLI, Anda harus menambahkan `lambda:InvokeFunction` pernyataan `lambda:InvokeFunctionUrl` dan secara terpisah. Contoh:

```
aws lambda add-permission --function-name my-function \
  --statement-id UrlPolicyInvokeURL \
  --action lambda:InvokeFunctionUrl \
  --principal 444455556666 \
  --function-url-auth-type AWS_IAM
```

```
aws lambda add-permission --function-name my-function \
  --statement-id UrlPolicyInvokeFunction \
  --action lambda:InvokeFunction \
  --principal 444455556666 \
  --invoked-via-function-url
```

Menggunakan tipe autentikasi **NONE**

> **Important**
>
> Ketika jenis autentikasi URL fungsi Anda NONE dan Anda memiliki kebijakan berbasis sumber daya yang memberikan akses publik, setiap pengguna yang tidak diautentikasi dengan URL fungsi Anda dapat memanggil fungsi Anda.

Dalam beberapa kasus, Anda mungkin ingin URL fungsi Anda menjadi publik. Misalnya, Anda mungkin ingin menyajikan permintaan yang dibuat langsung dari browser web. Untuk mengizinkan akses publik ke URL fungsi Anda, pilih jenis NONE autentikasi.

Jika Anda memilih jenis NONE autentikasi, Lambda tidak menggunakan IAM untuk mengautentikasi permintaan ke URL fungsi Anda. Namun, fungsi Anda harus memiliki kebijakan berbasis sumber daya yang memungkinkan dan. `lambda:InvokeFunctionUrl` `lambda:InvokeFunction` Saat Anda membuat URL fungsi dengan jenis autentikasi NONE menggunakan konsol atau AWS Serverless Application Model (AWS SAM), Lambda secara otomatis membuat kebijakan berbasis sumber daya untuk Anda. Jika Anda menggunakan AWS CLI, AWS CloudFormation, atau Lambda API secara langsung, Anda harus menambahkan sendiri kebijakan tersebut.

Sebaiknya sertakan kunci InvokedViaFunctionUrl konteks lambda: dalam kebijakan berbasis sumber daya saat menggunakan jenis autentikasi. NONE Kunci konteks ini memastikan bahwa fungsi hanya dapat dipanggil melalui URL fungsi dan tidak melalui metode pemanggilan lainnya.

Perhatikan hal berikut tentang kebijakan ini:

- Semua entitas dapat memanggil `lambda:InvokeFunctionUrl` dan`lambda:InvokeFunction`. Ini berarti bahwa siapa pun yang memiliki URL fungsi Anda dapat memanggil fungsi Anda.
- Nilai kunci `lambda:FunctionUrlAuthType` kondisi adalahNONE. Ini berarti bahwa pernyataan kebijakan mengizinkan akses hanya jika jenis autentikasi URL fungsi Anda jugaNONE.
- `lambda:InvokedViaFunctionUrl`Kondisi ini memastikan bahwa fungsi hanya dapat dipanggil melalui URL fungsi dan tidak melalui metode pemanggilan lainnya.

Example— Kebijakan berbasis sumber daya default untuk jenis autentikasi NONE

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "FunctionURLAllowPublicAccess",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "lambda:InvokeFunctionUrl",
      "Resource": "arn:aws:lambda:us-east-2:123456789012:function:my-function",
      "Condition": {
        "StringEquals": {
          "lambda:FunctionUrlAuthType": "NONE"
        }
      }
    },
    {
      "Sid": "FunctionURLInvokeAllowPublicAccess",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "lambda:InvokeFunction",
      "Resource": "arn:aws:lambda:us-east-2:123456789012:function:my-function",
      "Condition": {
        "Bool": {
          "lambda:InvokedViaFunctionUrl": "true"
        }
      }
    }
  ]
}
```

Buat kebijakan berbasis sumber daya menggunakan AWS CLI

Kecuali Anda menggunakan konsol atau AWS SAM untuk membuat URL fungsi dengan jenis autentikasi`NONE`, Anda harus menambahkan sendiri kebijakan berbasis sumber daya. Gunakan perintah berikut untuk membuat pernyataan untuk `lambda:InvokeFunctionUrl` dan `lambda:InvokeFunction` izin. Setiap pernyataan harus ditambahkan dalam perintah terpisah.

```
aws lambda add-permission \
  --function-name UrlTestFunction \
  --statement-id UrlPolicyInvokeURL \
  --action lambda:InvokeFunctionUrl \
  --principal * \
  --function-url-auth-type NONE
```

```
aws lambda add-permission \
  --function-name UrlTestFunction \
  --statement-id UrlPolicyInvokeFunction \
  --action lambda:InvokeFunction \
  --principal * \
  --invoked-via-function-url
```

> **Note**
>
> Jika Anda menghapus URL fungsi dengan jenis autentikasi`NONE`, Lambda tidak secara otomatis menghapus kebijakan berbasis sumber daya terkait. Jika ingin menghapus kebijakan ini, Anda harus melakukannya secara manual.

Jika kebijakan berbasis sumber daya fungsi tidak memberikan `lambda:invokeFunctionUrl` dan `lambda:InvokeFunction` izin, pengguna mendapatkan kode kesalahan 403 Forbidden saat mereka mencoba memanggil URL fungsi Anda. Ini terjadi bahkan jika URL fungsi menggunakan jenis `NONE` autentikasi.

Tata kelola dan kontrol akses

Selain izin pemanggilan URL fungsi, Anda juga dapat mengontrol akses pada tindakan yang digunakan untuk mengonfigurasi URL fungsi. Lambda mendukung tindakan kebijakan IAM berikut untuk URL fungsi:

- `lambda:InvokeFunctionUrl`— Memanggil Fungsi Peristiwa Lambda menggunakan URL fungsi.
- `lambda:CreateFunctionUrlConfig`— Buat URL fungsi dan atur`AuthType`.
- `lambda:UpdateFunctionUrlConfig`— Perbarui konfigurasi URL fungsi dan nya`AuthType`.
- `lambda:GetFunctionUrlConfig`— Lihat detail URL fungsi.
- `lambda:ListFunctionUrlConfigs`— Daftar konfigurasi URL fungsi.

- `lambda:DeleteFunctionUrlConfig`— Hapus URL fungsi.

Untuk mengizinkan atau menolak akses URL fungsi ke AWS entitas lain, sertakan tindakan ini dalam kebijakan IAM. Misalnya, kebijakan berikut memberikan `example` peran dalam Akun AWS `444455556666` izin untuk memperbarui URL fungsi untuk fungsi **`my-function`** di akun. `123456789012`

Example kebijakan URL fungsi lintas akun

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::444455556666:role/example"
            },
            "Action": "lambda:UpdateFunctionUrlConfig",
            "Resource": "arn:aws:lambda:us-east-2:123456789012:function:my-
function"
        }
    ]
}
```

Kunci syarat

Untuk kontrol akses berbutir halus atas URL fungsi Anda, gunakan tombol konteks kondisi. Lambda mendukung tombol konteks berikut untuk URL fungsi:

- `lambda:FunctionUrlAuthType`— Mendefinisikan nilai enum yang menjelaskan jenis autentikasi yang digunakan URL fungsi Anda. Nilai dapat berupa `AWS_IAM` atau `NONE`, salah satu.
- `lambda:InvokedViaFunctionUrl`— Membatasi `lambda:InvokeFunction` tindakan untuk panggilan yang dilakukan melalui URL fungsi. Ini memastikan bahwa fungsi hanya dapat dipanggil menggunakan URL fungsi dan tidak melalui metode pemanggilan lainnya. Untuk contoh kebijakan berbasis sumber daya yang menggunakan kunci `lambda:InvokedViaFunctionUrl` konteks, lihat contoh di dan. Menggunakan tipe autentikasi `AWS_IAM` Menggunakan tipe autentikasi `NONE`

Anda dapat menggunakan kunci konteks ini dalam kebijakan yang terkait dengan Fungsi Acara Anda. Misalnya, Anda mungkin ingin membatasi siapa yang dapat membuat perubahan konfigurasi pada URL fungsi Anda. Untuk menolak semua `UpdateFunctionUrlConfig` permintaan ke fungsi apa pun dengan jenis autentikasi URL`NONE`, Anda dapat menentukan kebijakan berikut:

Example kebijakan URL fungsi dengan penolakan eksplisit

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Principal": "*",
            "Action":[
                "lambda:UpdateFunctionUrlConfig"
            ],
            "Resource": "arn:aws:lambda:us-east-1:123456789012:function:*",
            "Condition": {
                "StringEquals": {
                    "lambda:FunctionUrlAuthType": "NONE"
                }
            }
        }
    ]
}
```

Untuk memberikan `example` peran dalam Akun AWS `444455556666` izin untuk membuat `CreateFunctionUrlConfig` dan `UpdateFunctionUrlConfig` permintaan pada fungsi dengan jenis autentikasi URL`AWS_IAM`, Anda dapat menentukan kebijakan berikut:

Example kebijakan URL fungsi dengan izin eksplisit

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
```

```
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::444455556666:role/example"
            },
            "Action":[
                "lambda:CreateFunctionUrlConfig",
                "lambda:UpdateFunctionUrlConfig"
            ],
            "Resource": "arn:aws:lambda:us-east-1:123456789012:function:*",
            "Condition": {
                "StringEquals": {
                    "lambda:FunctionUrlAuthType": "AWS_IAM"
                }
            }
        }
    ]
}
```

Anda juga dapat menggunakan kunci kondisi ini dalam kebijakan kontrol layanan (SCP). Gunakan SCP untuk mengelola izin di seluruh organisasi di. AWS Organizations Misalnya, untuk menolak pengguna membuat atau memperbarui URL fungsi yang menggunakan apa pun selain jenis `AWS_IAM` autentikasi, gunakan kebijakan kontrol layanan berikut:

Example fungsi URL SCP dengan penolakan eksplisit

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action":[
                "lambda:CreateFunctionUrlConfig",
                "lambda:UpdateFunctionUrlConfig"
            ],
            "Resource": "arn:aws:lambda:*:123456789012:function:*",
            "Condition": {
                "StringNotEquals": {
                    "lambda:FunctionUrlAuthType": "AWS_IAM"
                }
```

```
            }
        }
    ]
}
```

Memanggil URL fungsi Lambda

URL fungsi adalah titik akhir HTTP (S) khusus untuk Fungsi Acara Lambda Anda. Anda dapat membuat dan mengonfigurasi URL fungsi melalui konsol Lambda atau API Lambda.

> **Tip**
>
> Lambda menawarkan dua cara untuk menjalankan fungsi Anda melalui titik akhir HTTP: URL fungsi dan Amazon API Gateway. Jika Anda tidak yakin metode mana yang terbaik untuk kasus penggunaan Anda, lihatthe section called “URL Fungsi vs Amazon API Gateway”.

Saat Anda membuat URL fungsi, Lambda secara otomatis menghasilkan titik akhir URL unik untuk Anda. Setelah Anda membuat URL fungsi, titik akhir URL-nya tidak pernah berubah. Titik akhir URL fungsi memiliki format berikut:

```
https://<url-id>.lambda-url.<region>.on.aws
```

> **Note**
>
> URL fungsi tidak didukung di semua AWS wilayah. Untuk memeriksa ketersediaan, lihat Layanan AWS Regional dan filter berdasarkan`Feature Name = Function URLs`.

URL fungsi adalah dual stack-enabled, mendukung IPv4 dan IPv6. Setelah mengkonfigurasi URL fungsi Anda, Anda dapat memanggil fungsi Anda melalui titik akhir HTTP (S) dengan menggunakan browser web, curl, Postman, atau klien HTTP apa pun. Untuk memanggil URL fungsi, Anda harus memiliki `lambda:InvokeFunctionUrl` dan `lambda:InvokeFunction` izin. Untuk informasi selengkapnya, lihat Kontrol akses.

Topik

- Dasar-dasar pemanggilan URL fungsi
- Muatan permintaan dan respons

Dasar-dasar pemanggilan URL fungsi

Jika URL fungsi Anda menggunakan jenis `AWS_IAM` autentikasi, Anda harus menandatangani setiap permintaan HTTP menggunakan AWS Signature Version 4 (SigV4). Alat seperti Postman menawarkan cara bawaan untuk menandatangani permintaan Anda dengan SiGv4.

Jika Anda tidak menggunakan alat untuk menandatangani permintaan HTTP ke URL fungsi Anda, Anda harus menandatangani setiap permintaan secara manual menggunakan SigV4. Saat URL fungsi Anda menerima permintaan, Lambda juga menghitung tanda tangan SigV4. Lambda memproses permintaan hanya jika tanda tangan cocok. Untuk petunjuk tentang cara menandatangani permintaan Anda secara manual dengan SigV4, lihat Menandatangani AWS permintaan dengan Tanda Tangan Versi 4 di Referensi Umum Amazon Web Services Panduan.

Jika URL fungsi Anda menggunakan jenis `NONE` autentikasi, Anda tidak perlu menandatangani permintaan menggunakan SigV4. Anda dapat menjalankan fungsi Anda menggunakan browser web, curl, Postman, atau klien HTTP apa pun.

Untuk menguji `GET` permintaan sederhana ke fungsi Anda, gunakan browser web. Misalnya, jika URL fungsi Anda`https://abcdefg.lambda-url.us-east-1.on.aws`, dan dibutuhkan dalam parameter string`message`, URL permintaan Anda bisa terlihat seperti ini:

```
https://abcdefg.lambda-url.us-east-1.on.aws/?message=HelloWorld
```

Untuk menguji permintaan HTTP lainnya, seperti `POST` permintaan, Anda dapat menggunakan alat seperti curl. Misalnya, jika Anda ingin menyertakan beberapa data JSON dalam `POST` permintaan ke URL fungsi Anda, Anda dapat menggunakan perintah curl berikut:

```
curl -v 'https://abcdefg.lambda-url.us-east-1.on.aws/?message=HelloWorld' \
-H 'content-type: application/json' \
-d '{ "example": "test" }'
```

Muatan permintaan dan respons

Saat klien memanggil URL fungsi Anda, Lambda memetakan permintaan ke objek peristiwa sebelum meneruskannya ke fungsi Anda. Respons fungsi Anda kemudian dipetakan ke respons HTTP yang dikirim Lambda kembali ke klien melalui URL fungsi.

Format peristiwa permintaan dan respons mengikuti skema yang sama dengan format payload Amazon API Gateway versi 2.0.

Minta format muatan

Muatan permintaan memiliki struktur sebagai berikut:

```
{
  "version": "2.0",
  "routeKey": "$default",
  "rawPath": "/my/path",
  "rawQueryString": "parameter1=value1&parameter1=value2&parameter2=value",
  "cookies": [
    "cookie1",
    "cookie2"
  ],
  "headers": {
    "header1": "value1",
    "header2": "value1,value2"
  },
  "queryStringParameters": {
    "parameter1": "value1,value2",
    "parameter2": "value"
  },
  "requestContext": {
    "accountId": "123456789012",
    "apiId": "<urlid>",
    "authentication": null,
    "authorizer": {
        "iam": {
                "accessKey": "AKIA...",
                "accountId": "111122223333",
                "callerId": "AIDA...",
                "cognitoIdentity": null,
                "principalOrgId": null,
                "userArn": "arn:aws:iam::111122223333:user/example-user",
                "userId": "AIDA..."
        }
    },
    "domainName": "<url-id>.lambda-url.us-west-2.on.aws",
    "domainPrefix": "<url-id>",
    "http": {
      "method": "POST",
      "path": "/my/path",
      "protocol": "HTTP/1.1",
      "sourceIp": "123.123.123.123",
      "userAgent": "agent"
```

```
    },
    "requestId": "id",
    "routeKey": "$default",
    "stage": "$default",
    "time": "12/Mar/2020:19:03:58 +0000",
    "timeEpoch": 1583348638390
  },
  "body": "Hello from client!",
  "pathParameters": null,
  "isBase64Encoded": false,
  "stageVariables": null
}
```

Parameter	Deskripsi	Contoh
`version`	Versi format payload untuk acara ini. URL fungsi Lambda saat ini mendukung format payload versi 2.0.	`2.0`
`routeKey`	URL fungsi tidak menggunakan parameter ini. Lambda menetapkan ini `$default` sebagai placeholder.	`$default`
`rawPath`	Jalur permintaan. Misalnya, jika URL permintaan adalah`https://{url-id}.lambda-url.{region}.on.aws/example/test/demo`, maka nilai jalur mentah adalah`/example/test/demo`.	`/example/test/demo`
`rawQueryString`	String mentah yang berisi parameter string query permintaan. Karakter yang	`"?parameter1=value1&parameter2=value2"`

Parameter	Deskripsi	Contoh
	didukung termasuk a-zA-Z,0-9,.,_,-,%,,&,=, dan+.	
cookies	Array yang berisi semua cookie yang dikirim sebagai bagian dari permintaan.	["Cookie_1=Value_1", "Cookie_2=Value_2"]
headers	Daftar header permintaan, disajikan sebagai pasangan kunci-nilai.	{"header1": "value1", "header2": "value2"}
queryStringParameters	Parameter kueri untuk permintaan. Misalnya, jika URL permintaan adalahhttps://{url-id}.lambda-url.{region}.on.aws/example?name=Jane , maka queryStringParameters nilainya adalah objek JSON dengan kunci name dan nilai. Jane	{"name": "Jane"}
requestContext	Objek yang berisi informasi tambahan tentang permintaan, sepertirequestId , waktu permintaan, dan identitas penelepon jika diotorisasi melalui AWS Identity and Access Management (IAM).	
requestContext.accountId	Akun AWS ID pemilik fungsi.	"123456789012"
requestContext.apiId	ID URL fungsi.	"33anwqw8fj"

Parameter	Deskripsi	Contoh
`requestContext.authentication`	URL fungsi tidak menggunakan parameter ini. Lambda mengatur ini ke. `null`	`null`
`requestContext.authorizer`	Objek yang berisi informasi tentang identitas pemanggil, jika URL fungsi menggunakan jenis `AWS_IAM` autentikasi. Jika tidak, Lambda menyetel ini ke. `null`	
`requestContext.authorizer.iam.accessKey`	Kunci akses identitas penelepon.	`"AKIAIOSFODNN7EXAMPLE"`
`requestContext.authorizer.iam.accountId`	Akun AWS ID identitas penelepon.	`"111122223333"`
`requestContext.authorizer.iam.callerId`	ID (ID pengguna) penelepon.	`"AIDACKCEVSQ6C2EXAMPLE"`
`requestContext.authorizer.iam.cognitoIdentity`	URL fungsi tidak menggunakan parameter ini. Lambda menyetel ini ke `null` atau mengecualikan ini dari JSON.	`null`
`requestContext.authorizer.iam.principalOrgId`	ID organisasi utama yang terkait dengan identitas penelepon.	`"AIDACKCEVSQORGEXAMPLE"`
`requestContext.authorizer.iam.userArn`	Nama Sumber Daya Amazon (ARN) pengguna dari identitas penelepon.	`"arn:aws:iam::111122223333:user/example-user"`

Parameter	Deskripsi	Contoh
`requestContext.authorizer.iam.userId`	ID pengguna dari identitas pemanggil.	`"AIDACOSFODNN7EXAMPLE2"`
`requestContext.domainName`	Nama domain dari URL fungsi.	`"<url-id>.lambda-url.us-west-2.on.aws"`
`requestContext.domainPrefix`	Awalan domain dari URL fungsi.	`"<url-id>"`
`requestContext.http`	Objek yang berisi rincian tentang permintaan HTTP.	
`requestContext.http.method`	Metode HTTP yang digunakan dalam permintaan ini. Nilai yang valid termasuk `GET`, `POST`, `PUT`, `HEAD`, `OPTIONS`, `PATCH`, dan `DELETE`.	`GET`
`requestContext.http.path`	Jalur permintaan. Misalnya, jika URL permintaan adalah`https://{url-id}.lambda-url.{region}.on.aws/example/test/demo`, maka nilai jalurnya adalah`/example/test/demo`.	`/example/test/demo`
`requestContext.http.protocol`	Protokol permintaan.	`HTTP/1.1`
`requestContext.http.sourceIp`	Alamat IP sumber dari koneksi TCP langsung membuat permintaan.	`123.123.123.123`

Parameter	Deskripsi	Contoh
`requestContext.http.userAgent`	Nilai header User-Agent permintaan.	`Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) Gecko/20100101 Firefox/42.0`
`requestContext.requestId`	ID permintaan pemanggilan. Anda dapat menggunakan ID ini untuk melacak log pemanggilan yang terkait dengan fungsi Anda.	`e1506fd5-9e7b-434f-bd42-4f8fa224b599`
`requestContext.routeKey`	URL fungsi tidak menggunakan parameter ini. Lambda menetapkan ini `$default` sebagai placeholder.	`$default`
`requestContext.stage`	URL fungsi tidak menggunakan parameter ini. Lambda menetapkan ini `$default` sebagai placeholder.	`$default`
`requestContext.time`	Stempel waktu permintaan.	`"07/Sep/2021:22:50:22 +0000"`
`requestContext.timeEpoch`	Stempel waktu permintaan, dalam waktu zaman Unix.	`"1631055022677"`
body	Tubuh permintaan. Jika jenis konten permintaan adalah biner, badan dikodekan base64.	`{"key1": "value1", "key2": "value2"}`
`pathParameters`	URL fungsi tidak menggunakan parameter ini. Lambda menyetel ini ke `null` atau mengecualikan ini dari JSON.	`null`

Parameter	Deskripsi	Contoh
`isBase64Encoded`	`TRUE`jika tubuh adalah muatan biner dan dikodekan base64. `FALSE`jika tidak.	`FALSE`
`stageVariables`	URL fungsi tidak menggunak an parameter ini. Lambda menyetel ini ke `null` atau mengecualikan ini dari JSON.	`null`

Format payload respon

Saat fungsi Anda mengembalikan respons, Lambda mem-parsing respons dan mengubahnya menjadi respons HTTP. Muatan respons fungsi memiliki format berikut:

```
{
   "statusCode": 201,
    "headers": {
        "Content-Type": "application/json",
        "My-Custom-Header": "Custom Value"
    },
    "body": "{ \"message\": \"Hello, world!\" }",
    "cookies": [
        "Cookie_1=Value1; Expires=21 Oct 2021 07:48 GMT",
        "Cookie_2=Value2; Max-Age=78000"
    ],
    "isBase64Encoded": false
}
```

Lambda menyimpulkan format respons untuk Anda. Jika fungsi Anda mengembalikan JSON yang valid dan tidak mengembalikan `astatusCode`, Lambda mengasumsikan hal berikut:

- `statusCode` adalah `200`.

> **Note**
>
> Yang valid `statusCode` berada dalam kisaran 100 hingga 599.

- `content-type` adalah `application/json`.

- `body`adalah respon fungsi.
- `isBase64Encoded` adalah `false`.

Contoh berikut menunjukkan bagaimana output Fungsi Peristiwa Lambda Anda memetakan ke payload respons, dan bagaimana payload respons memetakan ke respons HTTP akhir. Saat klien memanggil URL fungsi Anda, mereka melihat respons HTTP.

Contoh output untuk respon string

Keluaran Fungsi Acara Lambda	Output respons yang ditafsirkan	Respons HTTP (apa yang dilihat klien)
`"Hello, world!"`	`{ "statusCode": 200, "body": "Hello, world!", "headers": { "content-type": "application/json" }, "isBase64Encoded": false }`	`HTTP/2 200 date: Wed, 08 Sep 2021 18:02:24 GMT content-type: application/json content-length: 15 "Hello, world!"`

Contoh keluaran untuk respons JSON

Keluaran Fungsi Acara Lambda	Output respons yang ditafsirkan	Respons HTTP (apa yang dilihat klien)
`{ "message": "Hello, world!" }`	`{ "statusCode": 200, "body": { "message": "Hello, world!" }, "headers": { "content-type": "application/json"`	`HTTP/2 200 date: Wed, 08 Sep 2021 18:02:24 GMT content-type: application/json content-length: 34 {`

Keluaran Fungsi Acara Lambda	Output respons yang ditafsirkan	Respons HTTP (apa yang dilihat klien)
	<pre> }, "isBase64Encoded": false }</pre>	<pre> "message": "Hello, world!" }</pre>

Contoh keluaran untuk respons kustom

Keluaran Fungsi Acara Lambda	Output respons yang ditafsirkan	Respons HTTP (apa yang dilihat klien)

<pre>{
   "statusCode": 201,
    "headers": {
        "Content-Type":
 "application/json",
        "My-Custom-
Header": "Custom Value"
    },
    "body": JSON.stri
ngify({
        "message":
 "Hello, world!"
    }),
    "isBase64Encoded":
 false
}</pre>
	<pre>{
	   "statusCode": 201,
	    "headers": {
	        "Content-Type":
	 "application/json",
	        "My-Custom-
	Header": "Custom Value"
	    },
	    "body": JSON.stri
	ngify({
	        "message":
	 "Hello, world!"
	    }),
	    "isBase64Encoded":
	 false
	}</pre>
		<pre>HTTP/2 201
		date: Wed, 08 Sep 2021
		 18:02:24 GMT
		content-type: applicati
		on/json
		content-length: 27
		my-custom-header:
		 Custom Value
		
		{
		  "message": "Hello,
		 world!"
		}</pre>

Cookie

Untuk mengembalikan cookie dari fungsi Anda, jangan menambahkan `set-cookie` header secara manual. Sebagai gantinya, sertakan cookie di objek payload respons Anda. Lambda secara otomatis menafsirkan ini dan menambahkannya sebagai `set-cookie` header dalam respons HTTP Anda, seperti pada contoh berikut.

Keluaran Fungsi Acara Lambda	Respons HTTP (apa yang dilihat klien)

```
{
   "statusCode": 201,
    "headers": {
        "Content-Type": "application/
json",
        "My-Custom-Header": "Custom
 Value"
    },
    "body": JSON.stringify({
        "message": "Hello, world!"
    }),
    "cookies": [
        "Cookie_1=Value1; Expires=21
 Oct 2021 07:48 GMT",
        "Cookie_2=Value2; Max-Age=7
8000"
    ],
    "isBase64Encoded": false
}
```

```
HTTP/2 201
date: Wed, 08 Sep 2021 18:02:24 GMT
content-type: application/json
content-length: 27
my-custom-header: Custom Value
set-cookie: Cookie_1=Value2;
 Expires=21 Oct 2021 07:48 GMT
set-cookie: Cookie_2=Value2; Max-
Age=78000

{
  "message": "Hello, world!"
}
```

Memantau fungsi Lambda URLs

Anda dapat menggunakan AWS CloudTrail dan Amazon CloudWatch untuk memantau fungsi Anda URLs.

Topik

- Fungsi pemantauan URLs dengan CloudTrail
- CloudWatch metrik untuk fungsi URLs

Fungsi pemantauan URLs dengan CloudTrail

Untuk fungsi URLs, Lambda secara otomatis mendukung pencatatan operasi API berikut sebagai peristiwa dalam file CloudTrail log:

- CreateFunctionUrlConfig
- UpdateFunctionUrlConfig
- DeleteFunctionUrlConfig
- GetFunctionUrlConfig
- ListFunctionUrlConfigs

Setiap entri log berisi informasi tentang identitas penelepon, kapan permintaan dibuat, dan detail lainnya. Anda dapat melihat semua acara dalam 90 hari terakhir dengan melihat riwayat CloudTrail Acara Anda. Untuk menyimpan catatan 90 hari terakhir, Anda dapat membuat jejak.

Secara default, CloudTrail tidak mencatat `InvokeFunctionUrl` permintaan, yang dianggap peristiwa data. Namun, Anda dapat mengaktifkan login peristiwa data CloudTrail. Untuk informasi lebih lanjut, lihat Peristiwa Pencatatan Data untuk Pelacakan dalam AWS CloudTrail Panduan Pengguna.

CloudWatch metrik untuk fungsi URLs

Lambda mengirimkan metrik agregat tentang permintaan URL fungsi ke. CloudWatch Dengan metrik ini, Anda dapat memantau fungsi URLs, membuat dasbor, dan mengonfigurasi alarm di konsol. CloudWatch

Fungsi URLs mendukung metrik pemanggilan berikut. Kami merekomendasikan untuk melihat metrik ini dengan `Sum` statistik.

- `UrlRequestCount`— Jumlah permintaan yang dibuat untuk URL fungsi ini.
- `Url4xxCount`— Jumlah permintaan yang mengembalikan kode status HTTP 4XX. Kode seri 4XX menunjukkan kesalahan sisi klien, seperti permintaan buruk.
- `Url5xxCount`— Jumlah permintaan yang mengembalikan kode status HTTP 5XX. Kode seri 5XX menunjukkan kesalahan sisi server, seperti kesalahan fungsi dan batas waktu.

Fungsi URLs juga mendukung metrik kinerja berikut. Kami merekomendasikan untuk melihat metrik ini dengan `Average` atau `Max` statistik.

- `UrlRequestLatency`— Waktu antara ketika URL fungsi menerima permintaan dan ketika URL fungsi mengembalikan respons.

Masing-masing metrik pemanggilan dan kinerja ini mendukung dimensi berikut:

- `FunctionName`— Lihat metrik agregat untuk fungsi yang URLs ditetapkan ke versi fungsi yang `$LATEST` tidak dipublikasikan, atau ke alias fungsi mana pun. Misalnya, `hello-world-function`.
- `Resource`— Lihat metrik untuk URL fungsi tertentu. Ini didefinisikan oleh nama fungsi, bersama dengan versi fungsi yang `$LATEST` tidak dipublikasikan atau salah satu alias fungsi. Misalnya, `hello-world-function:$LATEST`.
- `ExecutedVersion`— Lihat metrik untuk URL fungsi tertentu berdasarkan versi yang dieksekusi. Anda dapat menggunakan dimensi ini terutama untuk melacak URL fungsi yang ditetapkan ke versi yang `$LATEST` tidak dipublikasikan.

Pilih metode untuk memanggil Fungsi Peristiwa Lambda Anda menggunakan permintaan HTTP

Banyak kasus penggunaan umum untuk Lambda melibatkan pemanggilan Fungsi Acara Anda menggunakan permintaan HTTP. Misalnya, Anda mungkin ingin aplikasi web menjalankan fungsi Anda melalui permintaan browser. Fungsi Peristiwa Lambda juga dapat digunakan untuk membuat API REST lengkap, menangani interaksi pengguna dari aplikasi seluler, memproses data dari layanan eksternal melalui panggilan HTTP, atau membuat webhook khusus.

Bagian berikut menjelaskan pilihan Anda untuk memanggil Lambda melalui HTTP dan memberikan informasi untuk membantu Anda membuat keputusan yang tepat untuk kasus penggunaan khusus Anda.

Apa pilihan Anda saat memilih metode pemanggilan HTTP?

Lambda menawarkan dua metode utama untuk menjalankan fungsi menggunakan permintaan HTTP - URL fungsi dan API Gateway. Perbedaan utama antara kedua opsi ini adalah sebagai berikut:

- URL fungsi Lambda menyediakan titik akhir HTTP langsung yang sederhana untuk Fungsi Peristiwa Lambda. Mereka dioptimalkan untuk kesederhanaan dan efektivitas biaya dan menyediakan jalur tercepat untuk mengekspos Fungsi Peristiwa Lambda melalui HTTP.
- API Gateway adalah layanan yang lebih canggih untuk membangun API berfitur lengkap. API Gateway dioptimalkan untuk membangun dan mengelola API produksi dalam skala besar dan menyediakan alat komprehensif untuk keamanan, pemantauan, dan manajemen lalu lintas.

Rekomendasi jika Anda sudah mengetahui kebutuhan Anda

Jika Anda sudah jelas tentang kebutuhan Anda, berikut adalah rekomendasi dasar kami:

Kami merekomendasikan URL fungsi untuk aplikasi sederhana atau prototipe di mana Anda hanya memerlukan metode otentikasi dasar dan request/response penanganan dan di mana Anda ingin meminimalkan biaya dan kompleksitas.

API Gateway adalah pilihan yang lebih baik untuk aplikasi produksi dalam skala besar atau untuk kasus di mana Anda memerlukan fitur yang lebih canggih seperti dukungan OpenAPI Description, pilihan opsi otentikasi, nama domain khusus, atau request/response penanganan kaya termasuk throttling, caching, dan transformasi. request/response

Apa yang harus dipertimbangkan saat memilih metode untuk menjalankan Fungsi Acara Lambda Anda

Saat memilih antara URL fungsi dan API Gateway, Anda perlu mempertimbangkan faktor-faktor berikut:

- Kebutuhan otentikasi Anda, seperti apakah Anda memerlukan OAuth atau Amazon Cognito untuk mengautentikasi pengguna
- Persyaratan penskalaan Anda dan kompleksitas API yang ingin Anda terapkan
- Apakah Anda memerlukan fitur canggih seperti validasi permintaan dan request/response pemformatan
- Persyaratan pemantauan Anda
- Sasaran biaya Anda

Dengan memahami faktor-faktor ini, Anda dapat memilih opsi yang paling menyeimbangkan persyaratan keamanan, kompleksitas, dan biaya Anda.

Informasi berikut merangkum perbedaan utama antara kedua opsi.

Autentikasi

- URL fungsi menyediakan opsi otentikasi dasar melalui AWS Identity and Access Management (IAM). Anda dapat mengonfigurasi titik akhir Anda menjadi publik (tidak ada otentikasi) atau memerlukan otentikasi IAM. Dengan autentikasi IAM, Anda dapat menggunakan AWS kredensyal standar atau peran IAM untuk mengontrol akses. Meskipun mudah diatur, pendekatan ini memberikan opsi terbatas dibandingkan dengan metode otentikat lainnya.
- API Gateway menyediakan akses ke berbagai opsi otentikasi yang lebih komprehensif. Selain autentikasi IAM, Anda dapat menggunakan otorisasi Lambda (logika otentikasi khusus), kumpulan pengguna Amazon Cognito, dan alur. OAuth2.0 Fleksibilitas ini memungkinkan Anda menerapkan skema otentikasi yang kompleks, termasuk penyedia otentikasi pihak ketiga, otentikasi berbasis token, dan otentikasi multi-faktor.

Request/response penanganan

- URL fungsi menyediakan permintaan HTTP dasar dan penanganan respons. Mereka mendukung metode HTTP standar dan menyertakan dukungan berbagi sumber daya lintas asal (CORS) bawaan. Meskipun mereka dapat menangani muatan JSON dan parameter kueri secara alami, mereka tidak menawarkan transformasi permintaan atau kemampuan validasi. Penanganan respons sama mudahnya — klien menerima respons dari Fungsi Peristiwa Lambda Anda persis seperti Lambda mengembalikannya.
- API Gateway menyediakan kemampuan penanganan permintaan dan respons yang canggih. Anda dapat menentukan validator permintaan, mengubah permintaan dan tanggapan menggunakan templat pemetaan, menyiapkan request/response header, dan menerapkan cache respons. API Gateway juga mendukung payload biner dan nama domain khusus dan dapat memodifikasi tanggapan sebelum mencapai klien. Anda dapat mengatur model untuk request/response validasi dan transformasi menggunakan JSON Schema.

Penskalaan

- URL fungsi menskalakan secara langsung dengan batas konkurensi Fungsi Acara Lambda Anda dan menangani lonjakan lalu lintas dengan meningkatkan fungsi Anda hingga batas konkurensi

maksimum yang dikonfigurasi. Setelah batas itu tercapai, Lambda merespons permintaan tambahan dengan respons HTTP 429. Tidak ada mekanisme antrian bawaan, jadi penanganan penskalaan sepenuhnya bergantung pada konfigurasi Lambda Event Function Anda. Secara default, Lambda Event Functions memiliki batas 1.000 eksekusi bersamaan per. Wilayah AWS

- API Gateway menyediakan kemampuan penskalaan tambahan di atas penskalaan Lambda sendiri. Ini termasuk kontrol antrian permintaan dan pelambatan bawaan, memungkinkan Anda mengelola lonjakan lalu lintas dengan lebih anggun. API Gateway dapat menangani hingga 10.000 permintaan per detik per wilayah secara default, dengan kapasitas burst 5.000 permintaan per detik. Ini juga menyediakan alat untuk membatasi permintaan pada tingkat yang berbeda (API, panggung, atau metode) untuk melindungi backend Anda.

Memantau

- URL fungsi menawarkan pemantauan dasar melalui CloudWatch metrik Amazon, termasuk jumlah permintaan, latensi, dan tingkat kesalahan. Anda mendapatkan akses ke metrik dan log Lambda standar, yang menunjukkan permintaan mentah yang masuk ke fungsi Anda. Meskipun ini memberikan visibilitas operasional yang penting, metrik difokuskan terutama pada eksekusi fungsi.
- API Gateway menyediakan kemampuan pemantauan komprehensif termasuk metrik terperinci, pencatatan, dan opsi penelusuran. Anda dapat memantau panggilan API, latensi, tingkat kesalahan, dan hit/miss tingkat cache melalui CloudWatch. API Gateway juga terintegrasi dengan AWS X-Ray untuk penelusuran terdistribusi dan menyediakan format logging yang dapat disesuaikan.

Biaya

- URL fungsi mengikuti model penetapan harga Lambda standar — Anda hanya membayar untuk pemanggilan fungsi dan waktu komputasi. Tidak ada biaya tambahan untuk titik akhir URL itu sendiri. Ini menjadikannya pilihan hemat biaya untuk API sederhana atau aplikasi dengan lalu lintas rendah jika Anda tidak memerlukan fitur tambahan API Gateway.
- API Gateway menawarkan tingkat gratis yang mencakup satu juta panggilan API yang diterima untuk REST API dan satu juta panggilan API yang diterima untuk API HTTP. Setelah ini, API Gateway mengenakan biaya untuk panggilan API, transfer data, dan caching (jika diaktifkan). Lihat halaman harga API Gateway untuk memahami biaya kasus penggunaan Anda sendiri.

Fitur lainnya

- URL fungsi dirancang untuk kesederhanaan dan integrasi Lambda langsung. Mereka mendukung titik akhir HTTP dan HTTPS, menawarkan dukungan CORS bawaan, dan menyediakan titik akhir dual-stack (IPv4 dan IPv6). Meskipun mereka tidak memiliki fitur canggih, mereka unggul dalam skenario di mana Anda memerlukan cara cepat dan mudah untuk mengekspos Fungsi Peristiwa Lambda melalui HTTP.
- API Gateway mencakup banyak fitur tambahan seperti versi API, manajemen tahap, kunci API untuk paket penggunaan, dokumentasi API melalui, API Swagger/OpenAPI, WebSocket API pribadi dalam VPC, dan integrasi WAF untuk keamanan tambahan. Ini juga mendukung penerapan kenari, integrasi tiruan untuk pengujian, dan integrasi dengan yang lain di luar Lambda. Layanan AWS

Pilih metode untuk menjalankan Fungsi Peristiwa Lambda Anda

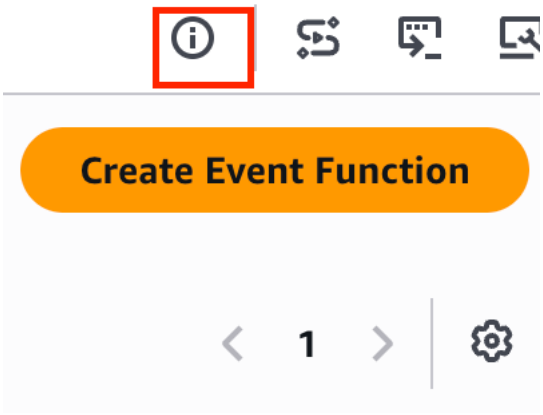
Sekarang setelah Anda membaca tentang kriteria untuk memilih antara URL fungsi Lambda dan API Gateway dan perbedaan utama di antara keduanya, Anda dapat memilih opsi yang paling sesuai dengan kebutuhan Anda dan menggunakan sumber daya berikut untuk membantu Anda mulai menggunakannya.

Function URLs

Memulai URL fungsi dengan sumber daya berikut

- Ikuti tutorial Membuat Fungsi Acara Lambda dengan URL fungsi
- Pelajari lebih lanjut tentang URL fungsi di the section called “URL fungsi” bagian panduan ini
- Coba tutorial yang dipandu dalam konsol Buat aplikasi web sederhana dengan melakukan hal berikut:

1. Buka halaman Fungsi Acara konsol Lambda.
2. Buka panel bantuan dengan memilih ikon di sudut kanan atas layar.



3. Pilih Tutorial.
4. Di Buat aplikasi web sederhana, pilih Mulai tutorial.

API Gateway

Memulai Lambda dan API Gateway dengan sumber daya berikut

- Ikuti tutorial Menggunakan Lambda dengan API Gateway untuk membuat REST API yang terintegrasi dengan Fungsi Acara Lambda backend.
- Pelajari selengkapnya tentang berbagai jenis API yang ditawarkan oleh API Gateway di bagian berikut dari Panduan Pengembang Amazon API Gateway:
  - API API Gateway REST
  - API API Gateway HTTP
  - API Gateway WebSocket API
- Coba satu atau beberapa contoh di bagian Tutorial dan lokakarya dari Panduan Pengembang Amazon API Gateway.

Tutorial: Membuat titik akhir webhook menggunakan URL fungsi Lambda

Dalam tutorial ini, Anda membuat URL fungsi Lambda untuk mengimplementasikan endpoint webhook. Webhook adalah komunikasi ringan yang digerakkan oleh peristiwa yang secara otomatis mengirimkan data antar aplikasi menggunakan HTTP. Anda dapat menggunakan webhook untuk menerima pembaruan langsung tentang peristiwa yang terjadi di sistem lain, seperti ketika pelanggan baru mendaftar di situs web, pembayaran diproses, atau file diunggah.

Dengan Lambda, webhook dapat diimplementasikan menggunakan URL fungsi Lambda atau API Gateway. URL fungsi adalah pilihan yang baik untuk webhook sederhana yang tidak memerlukan fitur seperti otorisasi lanjutan atau validasi permintaan.

> **Tip**
>
> Jika Anda tidak yakin solusi mana yang terbaik untuk kasus penggunaan khusus Anda, lihatthe section called “URL Fungsi vs Amazon API Gateway”.

Prasyarat

Untuk menyelesaikan tutorial ini, Anda harus memiliki Python (versi 3.8 atau yang lebih baru) atau Node.js (versi 18 atau lebih baru) diinstal pada mesin lokal Anda.

Untuk menguji titik akhir menggunakan permintaan HTTP, tutorial menggunakan curl, alat baris perintah yang dapat Anda gunakan untuk mentransfer data menggunakan berbagai protokol jaringan. Lihat dokumentasi curl untuk mempelajari cara menginstal alat jika Anda belum memilikinya.

Buat Fungsi Acara Lambda

Pertama buat Lambda Event Function yang berjalan ketika permintaan HTTP dikirim ke endpoint webhook Anda. Dalam contoh ini, aplikasi pengirim mengirimkan pembaruan setiap kali pembayaran dikirimkan dan menunjukkan di badan permintaan HTTP apakah pembayaran berhasil. Fungsi Acara Lambda mem-parsing permintaan dan mengambil tindakan sesuai dengan status pembayaran. Dalam contoh ini, kode hanya mencetak ID pesanan untuk pembayaran, tetapi dalam aplikasi nyata, Anda dapat menambahkan pesanan ke database atau mengirim pemberitahuan.

Fungsi ini juga mengimplementasikan metode otentikasi yang paling umum digunakan untuk webhook, otentikasi pesan berbasis hash (HMAC). Dengan metode ini, aplikasi pengirim dan penerima berbagi kunci rahasia. Aplikasi pengiriman menggunakan algoritma hashing untuk

menghasilkan tanda tangan unik menggunakan kunci ini bersama dengan konten pesan, dan menyertakan tanda tangan dalam permintaan webhook sebagai header HTTP. Aplikasi penerima kemudian mengulangi langkah ini, menghasilkan tanda tangan menggunakan kunci rahasia, dan membandingkan nilai yang dihasilkan dengan tanda tangan yang dikirim di header permintaan. Jika hasilnya cocok, permintaan dianggap sah.

Buat fungsi menggunakan konsol Lambda dengan Python atau runtime. Node.js

Python

Buat Fungsi Acara Lambda

1. Buka halaman Fungsi Acara konsol Lambda.
2. Buat fungsi dasar 'Hello world' dengan melakukan hal berikut:
   a. Pilih Buat Fungsi Acara.
   b. Pilih Penulis dari awal.
   c. Untuk Nama fungsi, masukkan **`myLambdaWebhook`**.
   d. Untuk Runtime, pilih python3.14.
   e. Pilih Buat Fungsi Acara.
3. Di panel Sumber kode, ganti kode yang ada dengan menyalin dan menempelkan berikut ini:

```
import json
import hmac
import hashlib
import os

def lambda_handler(event, context):

    # Get the webhook secret from environment variables
    webhook_secret = os.environ['WEBHOOK_SECRET']

    # Verify the webhook signature
    if not verify_signature(event, webhook_secret):
        return {
            'statusCode': 401,
            'body': json.dumps({'error': 'Invalid signature'})
        }

    try:
```

```
        # Parse the webhook payload
        payload = json.loads(event['body'])

        # Handle different event types
        event_type = payload.get('type')

        if event_type == 'payment.success':
            # Handle successful payment
            order_id = payload.get('orderId')
            print(f"Processing successful payment for order {order_id}")

            # Add your business logic here
            # For example, update database, send notifications, etc.

        elif event_type == 'payment.failed':
            # Handle failed payment
            order_id = payload.get('orderId')
            print(f"Processing failed payment for order {order_id}")

            # Add your business logic here

        else:
            print(f"Received unhandled event type: {event_type}")

        # Return success response
        return {
            'statusCode': 200,
            'body': json.dumps({'received': True})
        }

    except json.JSONDecodeError:
        return {
            'statusCode': 400,
            'body': json.dumps({'error': 'Invalid JSON payload'})
        }
    except Exception as e:
        print(f"Error processing webhook: {e}")
        return {
            'statusCode': 500,
            'body': json.dumps({'error': 'Internal server error'})
        }

def verify_signature(event, webhook_secret):
    """
```

```
    Verify the webhook signature using HMAC
    """
    try:
        # Get the signature from headers
        signature = event['headers'].get('x-webhook-signature')

        if not signature:
            print("Error: Missing webhook signature in headers")
            return False

        # Get the raw body (return an empty string if the body key doesn't
exist)
        body = event.get('body', '')

        # Create HMAC using the secret key
        expected_signature = hmac.new(
            webhook_secret.encode('utf-8'),
            body.encode('utf-8'),
            hashlib.sha256
        ).hexdigest()

        # Compare the expected signature with the received signature to
authenticate the message
        is_valid = hmac.compare_digest(signature, expected_signature)
        if not is_valid:
            print(f"Error: Invalid signature. Received: {signature}, Expected:
{expected_signature}")
            return False

        return True
    except Exception as e:
        print(f"Error verifying signature: {e}")
        return False
```

4. Di bagian DEPLOY, pilih Deploy untuk memperbarui kode fungsi Anda.

Node.js

Buat Fungsi Acara Lambda

1. Buka halaman Fungsi Acara konsol Lambda.
2. Buat fungsi dasar 'Hello world' dengan melakukan hal berikut:

a. Pilih Buat Fungsi Acara.

b. Pilih Penulis dari awal.

c. Untuk Nama fungsi, masukkan **myLambdaWebhook**.

d. Untuk Runtime, pilih nodejs24.x.

e. Pilih Buat Fungsi Acara.

3. Di panel Sumber kode, ganti kode yang ada dengan menyalin dan menempelkan berikut ini:

```
import crypto from 'crypto';

export const handler = async (event, context) => {
    // Get the webhook secret from environment variables
    const webhookSecret = process.env.WEBHOOK_SECRET;

    // Verify the webhook signature
    if (!verifySignature(event, webhookSecret)) {
        return {
            statusCode: 401,
            body: JSON.stringify({ error: 'Invalid signature' })
        };
    }

    try {
        // Parse the webhook payload
        const payload = JSON.parse(event.body);

        // Handle different event types
        const eventType = payload.type;

        switch (eventType) {
            case 'payment.success': {
                // Handle successful payment
                const orderId = payload.orderId;
                console.log(`Processing successful payment for order
 ${orderId}`);

                // Add your business logic here
                // For example, update database, send notifications, etc.
                break;
            }
```

```
            case 'payment.failed': {
                // Handle failed payment
                const orderId = payload.orderId;
                console.log(`Processing failed payment for order ${orderId}`);

                // Add your business logic here
                break;
            }

            default:
                console.log(`Received unhandled event type: ${eventType}`);
        }

        // Return success response
        return {
            statusCode: 200,
            body: JSON.stringify({ received: true })
        };

    } catch (error) {
        if (error instanceof SyntaxError) {
            // Handle JSON parsing errors
            return {
                statusCode: 400,
                body: JSON.stringify({ error: 'Invalid JSON payload' })
            };
        }

        // Handle all other errors
        console.error('Error processing webhook:', error);
        return {
            statusCode: 500,
            body: JSON.stringify({ error: 'Internal server error' })
        };
    }
};

// Verify the webhook signature using HMAC

const verifySignature = (event, webhookSecret) => {
    try {
        // Get the signature from headers
        const signature = event.headers['x-webhook-signature'];
```

```
        if (!signature) {
            console.log('No signature found in headers:', event.headers);
            return false;
        }

        // Get the raw body (return an empty string if the body key doesn't
exist)
        const body = event.body || '';

        // Create HMAC using the secret key
        const hmac = crypto.createHmac('sha256', webhookSecret);
        const expectedSignature = hmac.update(body).digest('hex');

        // Compare expected and received signatures
        const isValid = signature === expectedSignature;
        if (!isValid) {
            console.log(`Invalid signature. Received: ${signature}, Expected:
${expectedSignature}`);
            return false;
        }

        return true;
    } catch (error) {
        console.error('Error during signature verification:', error);
        return false;
    }
 };
```

4. Di bagian DEPLOY, pilih Deploy untuk memperbarui kode fungsi Anda.

Buat kunci rahasia

Untuk Lambda Event Function untuk mengotentikasi permintaan webhook, ia menggunakan kunci rahasia yang dibagikan dengan aplikasi panggilan. Dalam contoh ini, kunci disimpan dalam variabel lingkungan. Dalam aplikasi produksi, jangan sertakan informasi sensitif seperti kata sandi dalam kode fungsi Anda. Sebagai gantinya, buat AWS Secrets Manager rahasia dan kemudian gunakan ekstensi AWS Parameter dan Rahasia Lambda untuk mengambil kredensyal Anda di Fungsi Acara Lambda Anda.

Buat dan simpan kunci rahasia webhook

1. Hasilkan string acak yang panjang menggunakan generator nomor acak yang aman secara kriptografis. Anda dapat menggunakan cuplikan kode berikut dengan Python atau Node.js untuk menghasilkan dan mencetak rahasia 32 karakter, atau menggunakan metode pilihan Anda sendiri.

   Python

   Example kode untuk menghasilkan rahasia

   ```
   import secrets
   webhook_secret = secrets.token_urlsafe(32)
   print(webhook_secret)
   ```

   Node.js

   Example kode untuk menghasilkan rahasia (format modul ES)

   ```
   import crypto from 'crypto';
   let webhookSecret = crypto.randomBytes(32).toString('base64');
   console.log(webhookSecret)
   ```

2. Simpan string yang Anda hasilkan sebagai variabel lingkungan untuk fungsi Anda dengan melakukan hal berikut:

   a. Di tab Konfigurasi untuk fungsi Anda, pilih variabel Lingkungan.

   b. Pilih Edit.

   c. Pilih Tambahkan variabel lingkungan.

   d. Untuk Kunci, masukkan**`WEBHOOK_SECRET`**, lalu untuk Nilai, masukkan rahasia yang Anda buat di langkah sebelumnya.

   e. Pilih Simpan.

Anda harus menggunakan rahasia ini lagi nanti di tutorial untuk menguji fungsi Anda, jadi catat sekarang.

Buat titik akhir URL fungsi

Buat titik akhir untuk webhook Anda menggunakan URL fungsi Lambda. Karena Anda menggunakan jenis autentikasi `NONE` untuk membuat titik akhir dengan akses publik, siapa pun yang memiliki URL dapat memanggil fungsi Anda. Untuk mempelajari lebih lanjut tentang mengontrol akses ke URL fungsi, lihatthe section called “Kontrol akses”. Jika Anda memerlukan opsi otentikasi lanjutan untuk webhook Anda, pertimbangkan untuk menggunakan API Gateway.

Buat titik akhir URL fungsi

1. Di tab Konfigurasi untuk Fungsi Acara Anda, pilih URL Fungsi.
2. Pilih Buat URL fungsi.
3. Untuk jenis Auth, pilih NONE.
4. Pilih Simpan.

Titik akhir untuk URL fungsi yang baru saja Anda buat ditampilkan di panel URL Fungsi. Salin titik akhir untuk digunakan nanti dalam tutorial.

Uji Fungsi Acara di konsol

Sebelum menggunakan permintaan HTTP untuk memanggil Fungsi Acara Anda menggunakan titik akhir URL, uji di konsol untuk mengonfirmasi kode Anda berfungsi seperti yang diharapkan.

Untuk memverifikasi Fungsi Acara di konsol, pertama-tama Anda menghitung tanda tangan webhook menggunakan rahasia yang Anda buat sebelumnya dalam tutorial dengan payload JSON pengujian berikut:

```
{
    "type": "payment.success",
    "orderId": "1234",
    "amount": "99.99"
}
```

Gunakan salah satu contoh Python atau Node.js kode berikut untuk menghitung tanda tangan webhook menggunakan rahasia Anda sendiri.

Python

Hitung tanda tangan webhook

1. Simpan kode berikut sebagai file bernama`calculate_signature.py`. Ganti rahasia webhook dalam kode dengan nilai Anda sendiri.

```
import secrets
import hmac
import json
import hashlib

webhook_secret = "arlbSDCP86n_1H90s0fL_Qb2NAHBIBQOyGI0X4Zay4M"

body = json.dumps({"type": "payment.success", "orderId": "1234", "amount":
 "99.99"})

signature = hmac.new(
            webhook_secret.encode('utf-8'),
            body.encode('utf-8'),
            hashlib.sha256
        ).hexdigest()

print(signature)
```

2. Hitung tanda tangan dengan menjalankan perintah berikut dari direktori yang sama tempat Anda menyimpan kode. Salin tanda tangan output kode.

```
python calculate_signature.py
```

Node.js

Hitung tanda tangan webhook

1. Simpan kode berikut sebagai file bernama`calculate_signature.mjs`. Ganti rahasia webhook dalam kode dengan nilai Anda sendiri.

```
import crypto from 'crypto';

const webhookSecret = "arlbSDCP86n_1H90s0fL_Qb2NAHBIBQOyGI0X4Zay4M"
```

```
const body = "{\"type\": \"payment.success\", \"orderId\": \"1234\", \"amount\":
 \"99.99\"}";

let hmac = crypto.createHmac('sha256', webhookSecret);
let signature = hmac.update(body).digest('hex');

console.log(signature);
```

2. Hitung tanda tangan dengan menjalankan perintah berikut dari direktori yang sama tempat Anda menyimpan kode. Salin tanda tangan output kode.

```
node calculate_signature.mjs
```

Anda sekarang dapat menguji kode fungsi Anda menggunakan permintaan HTTP pengujian di konsol.

Uji Fungsi Acara di konsol

1. Pilih tab Kode untuk Fungsi Acara Anda.
2. Di bagian TEST EVENTS, pilih Create new test event
3. Untuk Nama Acara, masukkan**myEvent**.
4. Ganti JSON yang ada dengan menyalin dan menempelkan berikut ini ke panel Event JSON. Ganti tanda tangan webhook dengan nilai yang Anda hitung pada langkah sebelumnya.

```
{
  "headers": {
    "Content-Type": "application/json",
    "x-webhook-signature":
 "2d672e7a0423fab740fbc040e801d1241f2df32d2ffd8989617a599486553e2a"
  },
  "body": "{\"type\": \"payment.success\", \"orderId\": \"1234\", \"amount\":
 \"99.99\"}"
}
```

5. Pilih Simpan.
6. Pilih Panggil.

   Anda akan melihat output yang serupa dengan yang berikut:

Python

```
Status: Succeeded
Test Event Name: myEvent

Response:
{
  "statusCode": 200,
  "body": "{\"received\": true}"
}

Function Logs:
START RequestId: 50cc0788-d70e-453a-9a22-ceaa210e8ac6 Version: $LATEST
Processing successful payment for order 1234
END RequestId: 50cc0788-d70e-453a-9a22-ceaa210e8ac6
REPORT RequestId: 50cc0788-d70e-453a-9a22-ceaa210e8ac6 Duration: 1.55 ms Billed
 Duration: 2 ms Memory Size: 128 MB Max Memory Used: 36 MB Init Duration: 136.32
 ms
```

Node.js

```
Status: Succeeded
Test Event Name: myEvent

Response:
{
  "statusCode": 200,
  "body": "{\"received\":true}"
}

Function Logs:
START RequestId: e54fe6c7-1df9-4f05-a4c4-0f71cacd64f4 Version: $LATEST
2025-01-10T18:05:42.062Z e54fe6c7-1df9-4f05-a4c4-0f71cacd64f4 INFO Processing
 successful payment for order 1234
END RequestId: e54fe6c7-1df9-4f05-a4c4-0f71cacd64f4
REPORT RequestId: e54fe6c7-1df9-4f05-a4c4-0f71cacd64f4 Duration: 60.10 ms Billed
 Duration: 61 ms Memory Size: 128 MB Max Memory Used: 72 MB Init Duration:
 174.46 ms

Request ID: e54fe6c7-1df9-4f05-a4c4-0f71cacd64f4
```

Uji Fungsi Acara menggunakan permintaan HTTP

Gunakan alat baris perintah curl untuk menguji titik akhir webhook Anda.

Uji Fungsi Acara menggunakan permintaan HTTP

1. Dalam program terminal atau shell, jalankan perintah curl berikut. Ganti URL dengan nilai untuk titik akhir URL fungsi Anda sendiri dan ganti tanda tangan webhook dengan tanda tangan yang Anda hitung menggunakan kunci rahasia Anda sendiri.

   ```
   curl -X POST https://ryqgmbx5xjzxahif6frvzikpre0bpvpf.lambda-url.us-west-2.on.aws/
    \
   -H "Content-Type: application/json" \
   -H "x-webhook-
   signature: d5f52b76ffba65ff60ea73da67bdf1fc5825d4db56b5d3ffa0b64b7cb85ef48b" \
   -d '{"type": "payment.success", "orderId": "1234", "amount": "99.99"}'
   ```

   Anda akan melihat output berikut:

   ```
   {"received": true}
   ```

2. Periksa CloudWatch log untuk fungsi Anda untuk mengonfirmasinya mengurai muatan dengan benar dengan melakukan hal berikut:

   a. Buka halaman grup Log di CloudWatch konsol Amazon.

   b. Pilih grup log fungsi Anda (`/aws/lambda/myLambdaWebhook`).

   c. Pilih aliran log yang terbaru.

      Anda akan melihat output yang mirip dengan yang berikut di log fungsi Anda:

      Python

      ```
      Processing successful payment for order 1234
      ```

      Node.js

      ```
      2025-01-10T18:05:42.062Z e54fe6c7-1df9-4f05-a4c4-0f71cacd64f4 INFO
       Processing successful payment for order 1234
      ```

3. Konfirmasikan bahwa kode Anda mendeteksi tanda tangan yang tidak valid dengan menjalankan perintah curl berikut. Ganti URL dengan titik akhir URL fungsi Anda sendiri.

```
curl -X POST https://ryqgmbx5xjzxahif6frvzikpre0bpvpf.lambda-url.us-west-2.on.aws/
 \
-H "Content-Type: application/json" \
-H "x-webhook-signature: abcdefg" \
-d '{"type": "payment.success", "orderId": "1234", "amount": "99.99"}'
```

   Anda akan melihat output berikut:

```
{"error": "Invalid signature"}
```

Bersihkan sumber daya Anda

Sekarang Anda dapat menghapus sumber daya yang Anda buat untuk tutorial ini, kecuali Anda ingin mempertahankannya. Dengan menghapus AWS sumber daya yang tidak lagi Anda gunakan, Anda mencegah tagihan yang tidak perlu ke Anda Akun AWS.

Untuk menghapus Fungsi Acara Lambda

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih Fungsi Acara yang Anda buat.
3. Pilih Tindakan, Hapus.
4. Ketik **`confirm`** kolom input teks dan pilih Hapus.

Saat Anda membuat Fungsi Peristiwa Lambda di konsol, Lambda juga membuat peran eksekusi untuk fungsi Anda.

Untuk menghapus peran eksekusi

1. Buka halaman Peran dari konsol IAM.
2. Pilih peran eksekusi yang dibuat Lambda. Peran memiliki format nama`myLambdaWebhook-role-<random string>`.
3. Pilih Hapus.
4. Masukkan nama peran di bidang input teks dan pilih Hapus.

Memahami penskalaan Fungsi Acara Lambda

Konkurensi adalah jumlah permintaan dalam penerbangan yang ditangani oleh Fungsi AWS Lambda Acara Anda pada saat yang bersamaan. Untuk setiap permintaan bersamaan, Lambda menyediakan instance terpisah dari lingkungan eksekusi Anda. Saat Fungsi Acara Anda menerima lebih banyak permintaan, Lambda secara otomatis menangani penskalaan jumlah lingkungan eksekusi hingga Anda mencapai batas konkurensi akun Anda. Secara default, Lambda menyediakan akun Anda dengan batas konkurensi total 1.000 eksekusi bersamaan di semua Fungsi Acara dalam file. Wilayah AWS Untuk mendukung kebutuhan akun spesifik Anda, Anda dapat meminta peningkatan kuota dan mengonfigurasi kontrol konkurensi tingkat fungsi sehingga Fungsi Peristiwa penting Anda tidak mengalami pelambatan.

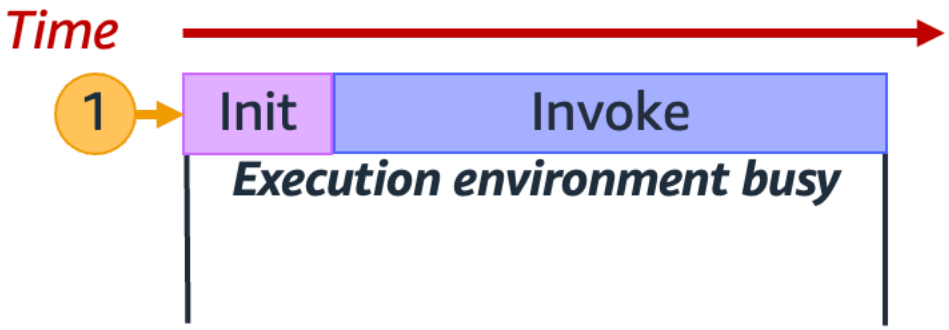
Topik ini menjelaskan konsep konkurensi dan penskalaan Fungsi Acara di Lambda. Pada akhir topik ini, Anda akan dapat memahami cara menghitung konkurensi, memvisualisasikan dua opsi kontrol konkurensi utama (dicadangkan dan disediakan), memperkirakan pengaturan kontrol konkurensi yang sesuai, dan melihat metrik untuk pengoptimalan lebih lanjut.

Bagian-bagian

- Memahami dan memvisualisasikan konkurensi
- Menghitung konkurensi untuk Fungsi Acara
- Memahami konkurensi cadangan dan konkurensi yang disediakan
- Memahami konkurensi dan permintaan per detik
- Kuota konkurensi
- Mengkonfigurasi konkurensi cadangan untuk Fungsi Acara
- Mengkonfigurasi konkurensi yang disediakan untuk Fungsi Peristiwa
- Perilaku penskalaan Fungsi Acara
- Memantau konkurensi

Memahami dan memvisualisasikan konkurensi

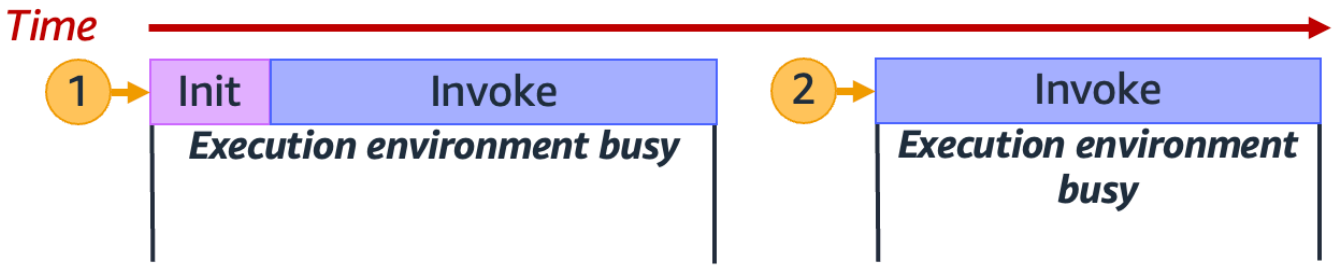
Lambda memanggil Fungsi Acara Anda di lingkungan eksekusi yang aman dan terisolasi. Untuk menangani permintaan, Lambda harus terlebih dahulu menginisialisasi lingkungan eksekusi (fase Init), sebelum menggunakannya untuk memanggil Fungsi Peristiwa Anda (fase Invoke):



> **Note**
>
> Durasi Init dan Invoke yang sebenarnya dapat bervariasi tergantung pada banyak faktor, seperti runtime yang Anda pilih dan kode Fungsi Peristiwa Lambda. Diagram sebelumnya tidak dimaksudkan untuk mewakili proporsi yang tepat dari durasi fase Init dan Invoke.

Diagram sebelumnya menggunakan persegi panjang untuk mewakili lingkungan eksekusi tunggal. Ketika Fungsi Acara Anda menerima permintaan pertamanya (diwakili oleh lingkaran kuning dengan label1), Lambda membuat lingkungan eksekusi baru dan menjalankan kode di luar handler utama Anda selama fase Init. Kemudian, Lambda menjalankan kode handler utama Event Function Anda selama fase Invoke. Selama seluruh proses ini, lingkungan eksekusi ini sibuk dan tidak dapat memproses permintaan lainnya.

Ketika Lambda selesai memproses permintaan pertama, lingkungan eksekusi ini kemudian dapat memproses permintaan tambahan untuk Fungsi Acara yang sama. Untuk permintaan berikutnya, Lambda tidak perlu menginisialisasi ulang lingkungan.

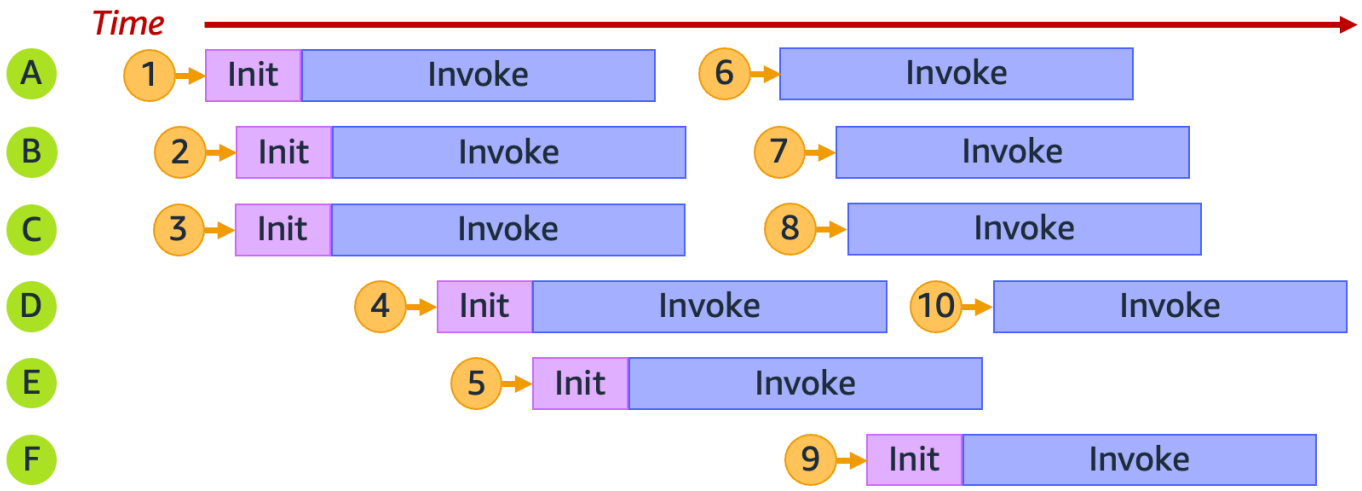


Pada diagram sebelumnya, Lambda menggunakan kembali lingkungan eksekusi untuk menangani permintaan kedua (diwakili oleh lingkaran kuning dengan label). 2

Sejauh ini, kami hanya berfokus pada satu contoh lingkungan eksekusi Anda (yaitu, konkurensi 1). Dalam praktiknya, Lambda mungkin perlu menyediakan beberapa instance lingkungan eksekusi secara paralel untuk menangani semua permintaan yang masuk. Ketika Fungsi Acara Anda menerima permintaan baru, salah satu dari dua hal dapat terjadi:

- Jika instance lingkungan eksekusi pra-inisialisasi tersedia, Lambda menggunakannya untuk memproses permintaan.
- Jika tidak, Lambda membuat instance lingkungan eksekusi baru untuk memproses permintaan.

Misalnya, mari kita jelajahi apa yang terjadi ketika Fungsi Acara Anda menerima 10 permintaan:



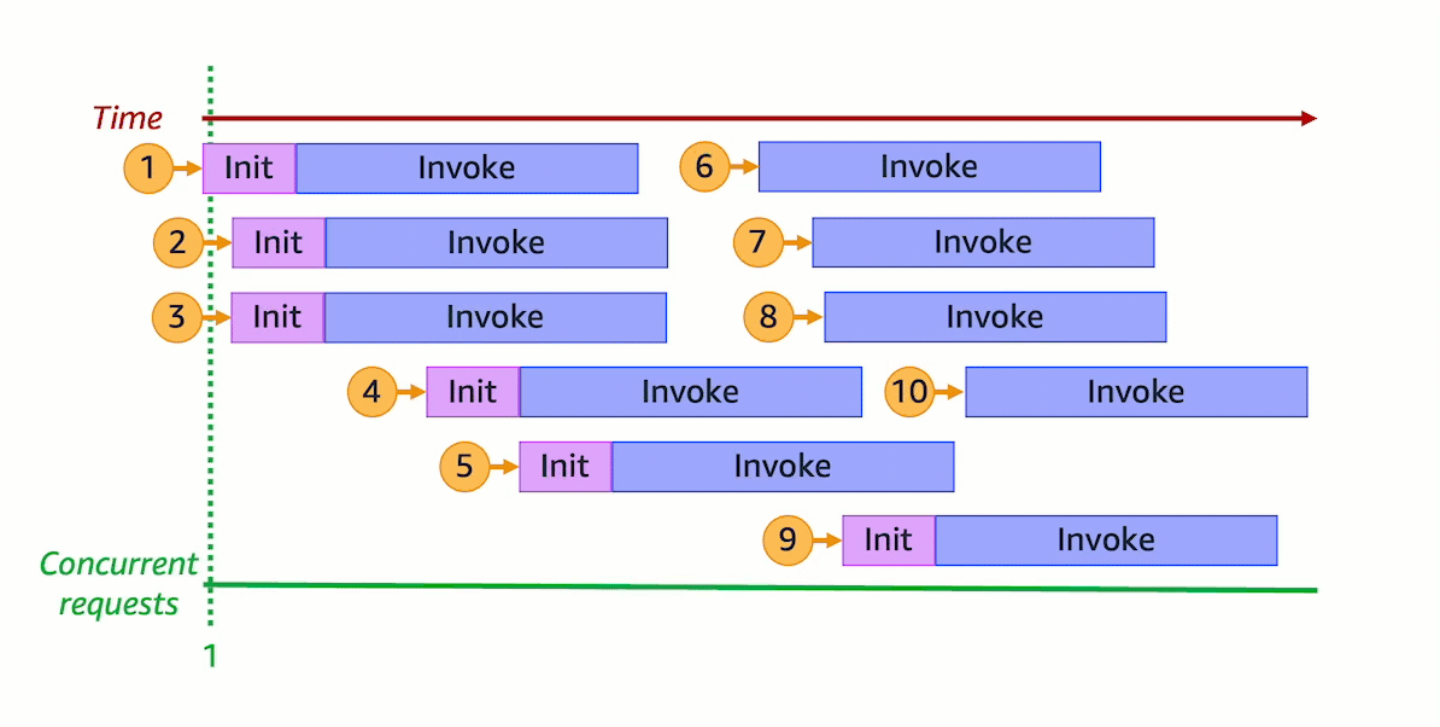
Dalam diagram sebelumnya, setiap bidang horizontal mewakili contoh lingkungan eksekusi tunggal (diberi label dari A melaluiF). Inilah cara Lambda menangani setiap permintaan:

Permintaan	Perilaku Lambda	Penalaran
1	Ketentuan lingkungan baru A	Ini adalah permintaan pertama; tidak ada instance lingkungan eksekusi yang tersedia.

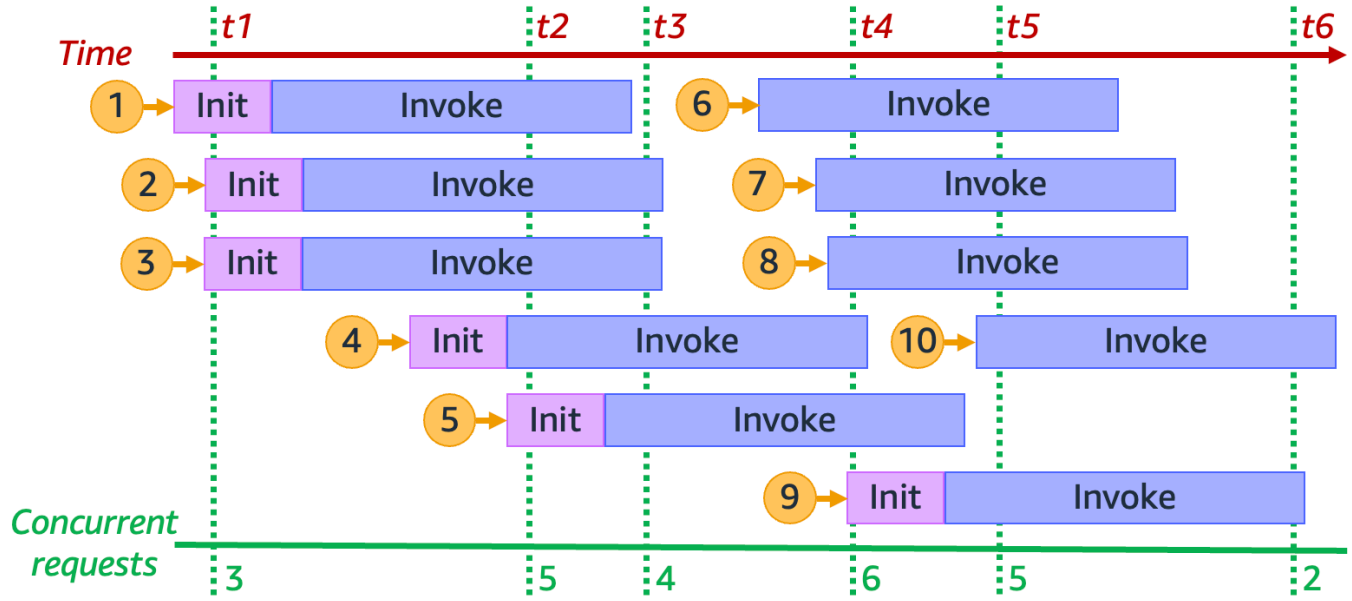
Permintaan	Perilaku Lambda	Penalaran
2	Ketentuan lingkungan baru B	Lingkungan eksekusi yang ada misalnya A sibuk.
3	Ketentuan lingkungan baru C	Instans lingkungan eksekusi yang ada A dan B keduanya sibuk.
4	Ketentuan lingkungan baru D	Instans lingkungan eksekusi yang ada A, B, dan C semuanya sibuk.
5	Ketentuan lingkungan baru E	Contoh lingkungan eksekusi yang ada A, B, C, dan D semuanya sibuk.
6	Menggunakan kembali lingkungan A	Lingkungan eksekusi instance A telah selesai memproses permintaan 1 dan sekarang tersedia.
7	Menggunakan kembali lingkungan B	Contoh lingkungan eksekusi B telah selesai memproses permintaan 2 dan sekarang tersedia.
8	Menggunakan kembali lingkungan C	Contoh lingkungan eksekusi C telah selesai memproses permintaan 3 dan sekarang tersedia.
9	Ketentuan lingkungan baru F	Contoh lingkungan eksekusi yang ada A, B, C, D, dan E semuanya sibuk.

Permintaan	Perilaku Lambda	Penalaran
10	Menggunakan kembali lingkungan D	Contoh lingkungan eksekusi D telah selesai memproses permintaan 4 dan sekarang tersedia.

Saat Fungsi Acara Anda menerima lebih banyak permintaan bersamaan, Lambda meningkatkan jumlah instance lingkungan eksekusi sebagai respons. Animasi berikut melacak jumlah permintaan bersamaan dari waktu ke waktu:



Dengan membekukan animasi sebelumnya pada enam titik waktu yang berbeda, kita mendapatkan diagram berikut:



Pada diagram sebelumnya, kita dapat menggambar garis vertikal kapan saja dan menghitung jumlah lingkungan yang memotong garis ini. Ini memberi kita jumlah permintaan bersamaan pada saat itu. Misalnya, pada waktu`t1`, ada tiga lingkungan aktif yang melayani tiga permintaan bersamaan. Jumlah maksimum permintaan bersamaan dalam simulasi ini terjadi pada waktu`t4`, ketika ada enam lingkungan aktif yang melayani enam permintaan bersamaan.

Untuk meringkas, konkurensi Fungsi Acara Anda adalah jumlah permintaan bersamaan yang ditangani pada saat yang sama. Menanggapi peningkatan konkurensi Fungsi Acara Anda, Lambda menyediakan lebih banyak instance lingkungan eksekusi untuk memenuhi permintaan permintaan.

Menghitung konkurensi untuk Fungsi Acara

Secara umum, konkurensi suatu sistem adalah kemampuan untuk memproses lebih dari satu tugas secara bersamaan. Di Lambda, konkurensi adalah jumlah permintaan dalam penerbangan yang ditangani Fungsi Acara Anda secara bersamaan. Cara cepat dan praktis untuk mengukur konkurensi Fungsi Peristiwa Lambda adalah dengan menggunakan rumus berikut:

```
Concurrency = (average requests per second) * (average request duration in seconds)
```

Konkurensi berbeda dari permintaan per detik. Misalnya, Fungsi Acara Anda menerima rata-rata 100 permintaan per detik. Jika durasi permintaan rata-rata adalah satu detik, maka memang benar bahwa konkurensi juga 100:

```
Concurrency = (100 requests/second) * (1 second/request) = 100
```

Namun, jika durasi permintaan rata-rata adalah 500 ms, maka konkurensi adalah 50:

```
Concurrency = (100 requests/second) * (0.5 second/request) = 50
```

Apa arti konkurensi 50 dalam praktik? Jika durasi permintaan rata-rata adalah 500 ms, maka Anda dapat menganggap instance fungsi Anda mampu menangani dua permintaan per detik. Kemudian, dibutuhkan 50 instance fungsi Anda untuk menangani beban 100 permintaan per detik. Konkurensi 50 berarti Lambda harus menyediakan 50 instance lingkungan eksekusi untuk menangani beban kerja ini secara efisien tanpa pembatasan apa pun. Berikut cara mengekspresikan ini dalam bentuk persamaan:

```
Concurrency = (100 requests/second) / (2 requests/second) = 50
```

Jika Fungsi Acara Anda menerima dua kali lipat jumlah permintaan (200 permintaan per detik), tetapi hanya membutuhkan separuh waktu untuk memproses setiap permintaan (250 ms), maka konkurensi masih 50:

```
Concurrency = (200 requests/second) * (0.25 second/request) = 50
```

Uji pemahaman Anda tentang konkurensi

Misalkan Anda memiliki Fungsi Acara yang membutuhkan, rata-rata, 200 ms untuk dijalankan. Selama beban puncak, Anda mengamati 5.000 permintaan per detik. Apa konkurensi Fungsi Acara Anda selama beban puncak?

Jawaban

Durasi Fungsi Acara rata-rata adalah 200 ms, atau 0,2 detik. Dengan menggunakan rumus konkurensi, Anda dapat memasukkan angka untuk mendapatkan konkurensi 1.000:

```
Concurrency = (5,000 requests/second) * (0.2 seconds/request) = 1,000
```

Atau, durasi Fungsi Acara rata-rata 200 ms berarti bahwa Fungsi Acara Anda dapat memproses 5 permintaan per detik. Untuk menangani beban kerja 5.000 permintaan per detik, Anda memerlukan 1.000 instance lingkungan eksekusi. Jadi, konkurensi adalah 1.000:

```
Concurrency = (5,000 requests/second) / (5 requests/second) = 1,000
```

Memahami konkurensi cadangan dan konkurensi yang disediakan

Secara default, akun Anda memiliki batas konkurensi 1.000 eksekusi bersamaan di semua Fungsi Peristiwa di Wilayah. Fungsi Acara Anda membagikan kumpulan 1.000 konkurensi ini berdasarkan permintaan. Fungsi Acara Anda mengalami pelambatan (yaitu, mereka mulai menghapus permintaan) jika Anda kehabisan konkurensi yang tersedia.

Beberapa Fungsi Acara Anda mungkin lebih penting daripada yang lain. Akibatnya, Anda mungkin ingin mengonfigurasi pengaturan konkurensi untuk memastikan bahwa Fungsi Peristiwa kritis mendapatkan konkurensi yang mereka butuhkan. Ada dua jenis kontrol konkurensi yang tersedia: konkurensi cadangan dan konkurensi yang disediakan.

- Gunakan konkurensi cadangan untuk mengatur jumlah maksimum dan minimum instans bersamaan untuk memesan sebagian konkurensi akun Anda untuk Fungsi Acara. Ini berguna jika Anda tidak ingin Fungsi Acara lain mengambil semua konkurensi tanpa syarat yang tersedia. Ketika Fungsi Peristiwa telah mencadangkan konkurensi, tidak ada Fungsi Peristiwa lain yang dapat menggunakan konkurensi itu.
- Gunakan konkurensi yang disediakan untuk menginisialisasi sejumlah instance lingkungan untuk Fungsi Peristiwa. Ini berguna untuk mengurangi latensi start dingin.

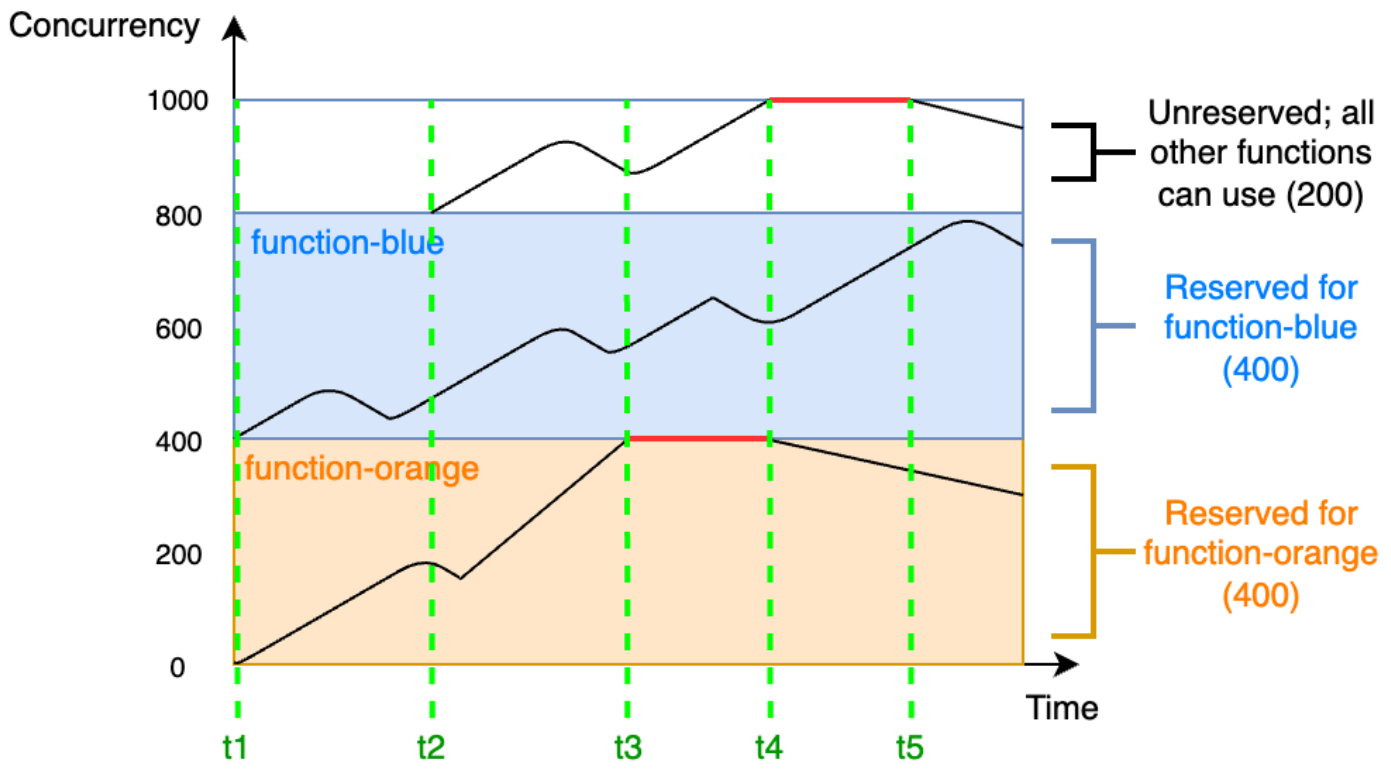
Konkurensi terpesan

Jika Anda ingin menjamin bahwa sejumlah konkurensi tersedia untuk Fungsi Acara Anda kapan saja, gunakan konkurensi cadangan.

Konkurensi cadangan menetapkan jumlah maksimum dan minimum instance bersamaan yang ingin Anda alokasikan ke Fungsi Acara Anda. Saat Anda mendedikasikan konkurensi cadangan ke Fungsi Acara, tidak ada Fungsi Acara lain yang dapat menggunakan konkurensi itu. Dengan kata lain, pengaturan konkurensi cadangan dapat memengaruhi kumpulan konkurensi yang tersedia untuk Fungsi Acara lainnya. Fungsi Acara yang tidak memiliki konkurensi cadangan berbagi kumpulan konkurensi tanpa syarat yang tersisa.

Mengkonfigurasi jumlah konkurensi cadangan terhadap batas konkurensi akun Anda secara keseluruhan. Tidak ada biaya untuk mengonfigurasi konkurensi cadangan untuk Fungsi Acara.

Untuk lebih memahami konkurensi cadangan, pertimbangkan diagram berikut:



Dalam diagram ini, batas konkurensi akun Anda untuk semua Fungsi Acara di Wilayah ini berada pada batas default 1.000. Misalkan Anda memiliki dua Fungsi Peristiwa penting, `function-blue` dan`function-orange`, yang secara rutin mengharapkan untuk mendapatkan volume pemanggilan yang tinggi. Anda memutuskan untuk memberikan 400 unit konkurensi cadangan ke`function-blue`, dan 400 unit konkurensi cadangan untuk. `function-orange` Dalam contoh ini, semua Fungsi Acara lainnya di akun Anda harus membagikan 200 unit konkurensi tanpa syarat yang tersisa.

Diagram memiliki lima poin menarik:

- Dit1, keduanya `function-orange` dan `function-blue` mulai menerima permintaan. Setiap Fungsi Acara mulai menggunakan bagian yang dialokasikan dari unit konkurensi cadangan.
- Dit2, `function-orange` dan `function-blue` terus menerima lebih banyak permintaan. Pada saat yang sama, Anda menerapkan beberapa Fungsi Peristiwa Lambda lainnya, yang mulai menerima permintaan. Anda tidak mengalokasikan konkurensi cadangan ke Fungsi Acara lainnya ini. Mereka mulai menggunakan 200 unit konkurensi tanpa syarat yang tersisa.
- Padat3, `function-orange` mencapai konkurensi maks 400. Meskipun ada konkurensi yang tidak digunakan di tempat lain di akun Anda, `function-orange` tidak dapat mengaksesnya. Garis

merah menunjukkan bahwa `function-orange` sedang mengalami pelambatan, dan Lambda mungkin membatalkan permintaan.

- Pada`t4`, `function-orange` mulai menerima lebih sedikit permintaan dan tidak lagi melambat. Namun, Fungsi Acara Anda yang lain mengalami lonjakan lalu lintas dan mulai melambat. Meskipun ada konkurensi yang tidak digunakan di tempat lain di akun Anda, Fungsi Acara lainnya ini tidak dapat mengaksesnya. Garis merah menunjukkan bahwa Fungsi Acara Anda yang lain mengalami pelambatan.
- Pada`t5`, Fungsi Acara lainnya mulai menerima lebih sedikit permintaan dan tidak lagi melambat.

Dari contoh ini, perhatikan bahwa reserving concurrency memiliki efek sebagai berikut:

- Fungsi Acara Anda dapat menskalakan secara independen dari Fungsi Acara lainnya di akun Anda. Semua Fungsi Acara akun Anda di Wilayah yang sama yang tidak memiliki konkurensi cadangan berbagi kumpulan konkurensi tanpa syarat. Tanpa konkurensi cadangan, Fungsi Acara lainnya berpotensi menggunakan semua konkurensi Anda yang tersedia. Ini mencegah Fungsi Peristiwa penting dari peningkatan skala jika diperlukan.
- Fungsi Acara Anda tidak dapat skala di luar kendali. Konkurensi cadangan membatasi konkurensi maksimum dan minimum Fungsi Acara Anda. Ini berarti bahwa Fungsi Acara Anda tidak dapat menggunakan konkurensi yang dicadangkan untuk Fungsi Acara lainnya, atau konkurensi dari kumpulan tanpa syarat. Selain itu, konkurensi cadangan bertindak sebagai batas bawah dan atas - ini menyimpan kapasitas yang ditentukan secara eksklusif untuk Fungsi Acara Anda sementara juga mencegahnya dari penskalaan melampaui batas itu. Anda dapat memesan konkurensi untuk mencegah Fungsi Acara Anda menggunakan semua konkurensi yang tersedia di akun Anda, atau dari kelebihan sumber daya hilir.
- Anda mungkin tidak dapat menggunakan semua konkurensi akun yang tersedia. Reservasi konkurensi diperhitungkan terhadap batas konkurensi akun Anda, tetapi ini juga berarti bahwa Fungsi Acara lainnya tidak dapat menggunakan potongan konkurensi cadangan tersebut. Jika Fungsi Acara Anda tidak menggunakan semua konkurensi yang Anda pesan untuk itu, Anda secara efektif membuang-buang konkurensi itu. Ini bukan masalah kecuali Fungsi Acara lain di akun Anda bisa mendapatkan keuntungan dari konkurensi yang terbuang.

Untuk mempelajari cara mengelola setelan konkurensi cadangan untuk Fungsi Acara Anda, lihat[Mengkonfigurasi konkurensi cadangan untuk Fungsi Acara](#).

Konkurensi yang disediakan

Anda menggunakan konkurensi cadangan untuk menentukan jumlah maksimum lingkungan eksekusi yang dicadangkan untuk Fungsi Peristiwa Lambda. Namun, tidak satu pun dari lingkungan ini yang diinisialisasi sebelumnya. Akibatnya, pemanggilan Fungsi Acara Anda mungkin memakan waktu lebih lama karena Lambda harus terlebih dahulu menginisialisasi lingkungan baru sebelum dapat menggunakannya untuk memanggil Fungsi Acara Anda. Ketika Lambda harus menginisialisasi lingkungan baru untuk melakukan doa, ini dikenal sebagai awal yang dingin. Untuk mengurangi start dingin, Anda dapat menggunakan konkurensi yang disediakan.

Konkurensi yang disediakan adalah jumlah lingkungan eksekusi pra-inisialisasi yang ingin Anda alokasikan ke Fungsi Acara Anda. Jika Anda menetapkan konkurensi yang disediakan pada Fungsi Peristiwa, Lambda menginisialisasi jumlah lingkungan eksekusi tersebut sehingga mereka siap untuk segera merespons permintaan Fungsi Acara.

> **Note**
>
> Menggunakan konkurensi yang disediakan menimbulkan biaya tambahan ke akun Anda. Jika Anda bekerja dengan runtime Java 11 atau Java 17, Anda juga dapat menggunakan SnapStart Lambda untuk mengurangi masalah start dingin tanpa biaya tambahan. SnapStart menggunakan snapshot cache dari lingkungan eksekusi Anda untuk meningkatkan kinerja startup secara signifikan. Anda tidak dapat menggunakan keduanya SnapStart dan konkurensi yang disediakan pada versi Fungsi Peristiwa yang sama. Untuk informasi selengkapnya tentang SnapStart fitur, batasan, dan Wilayah yang didukung, lihatMeningkatkan kinerja startup dengan Lambda SnapStart.

Saat menggunakan konkurensi yang disediakan, Lambda masih mendaur ulang lingkungan eksekusi di latar belakang. Misalnya, ini dapat terjadi setelah kegagalan pemanggilan. Namun, pada waktu tertentu, Lambda selalu memastikan bahwa jumlah lingkungan pra-inisialisasi sama dengan nilai pengaturan konkurensi yang disediakan Fungsi Acara Anda. Yang penting, bahkan jika Anda menggunakan konkurensi yang disediakan, Anda masih dapat mengalami penundaan awal yang dingin jika Lambda harus mengatur ulang lingkungan eksekusi.

Sebaliknya, saat menggunakan konkurensi cadangan, Lambda mungkin sepenuhnya mengakhiri lingkungan setelah periode tidak aktif. Diagram berikut mengilustrasikan hal ini dengan membandingkan siklus hidup lingkungan eksekusi tunggal saat Anda mengonfigurasi Fungsi Acara menggunakan konkurensi cadangan dibandingkan dengan konkurensi yang disediakan.

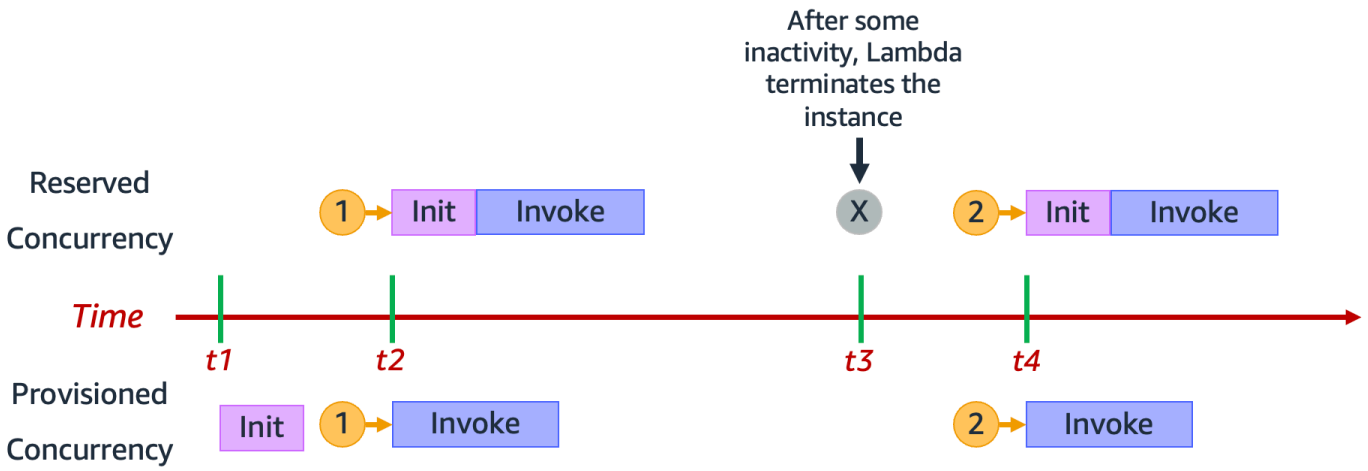
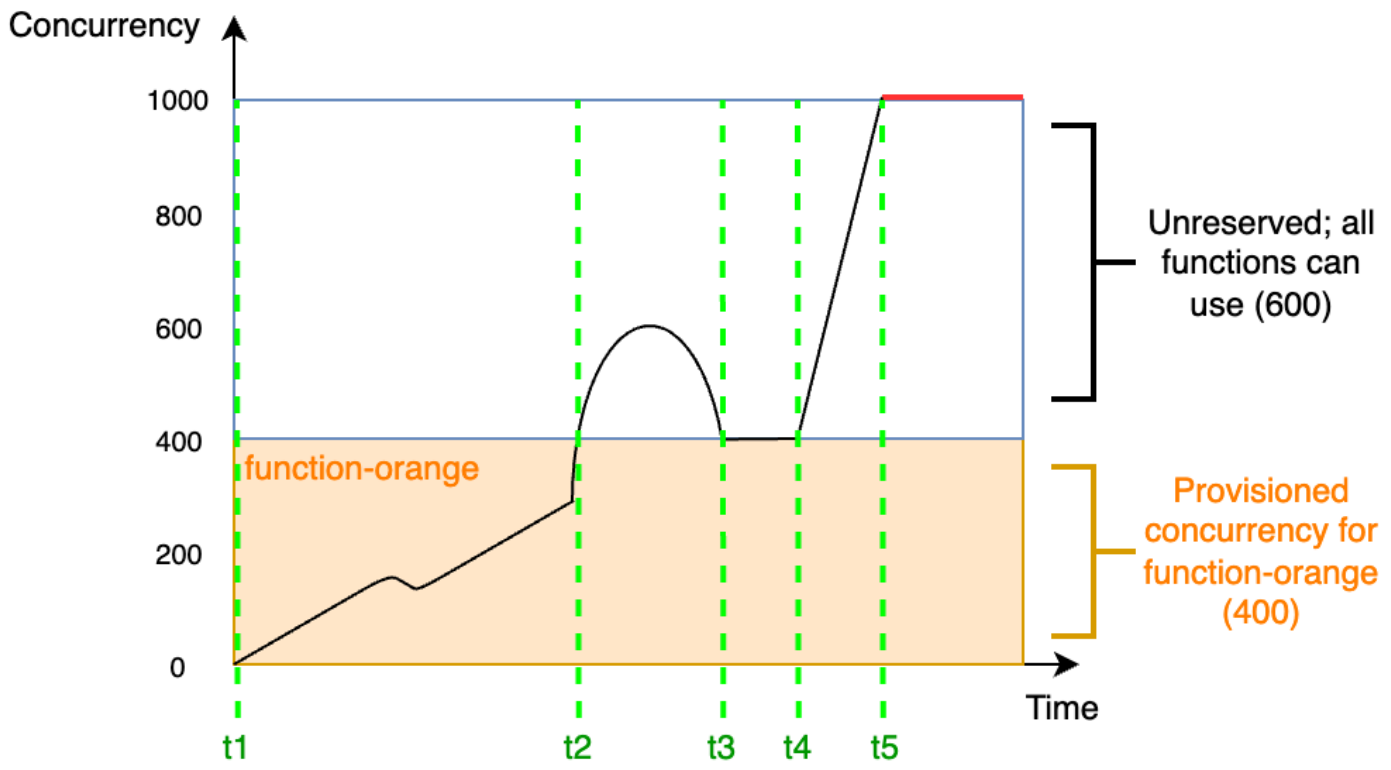


Diagram memiliki empat tempat menarik:

Waktu	Konkurensi terpesan	Konkurensi yang disediakan
t1	Tidak ada yang terjadi.	Lambda melakukan pra-inisi alisasi instance lingkungan eksekusi.
t2	Permintaan 1 masuk. Lambda harus menginisialisasi instance lingkungan eksekusi baru.	Permintaan 1 masuk. Lambda menggunakan instance lingkungan pra-inisialisasi.
t3	Setelah beberapa tidak aktif, Lambda menghentikan instance lingkungan aktif.	Tidak ada yang terjadi.
t4	Permintaan 2 masuk. Lambda harus menginisialisasi instance lingkungan eksekusi baru.	Permintaan 2 masuk. Lambda menggunakan instance lingkungan pra-inisialisasi.

Untuk lebih memahami konkurensi yang disediakan, pertimbangkan diagram berikut:



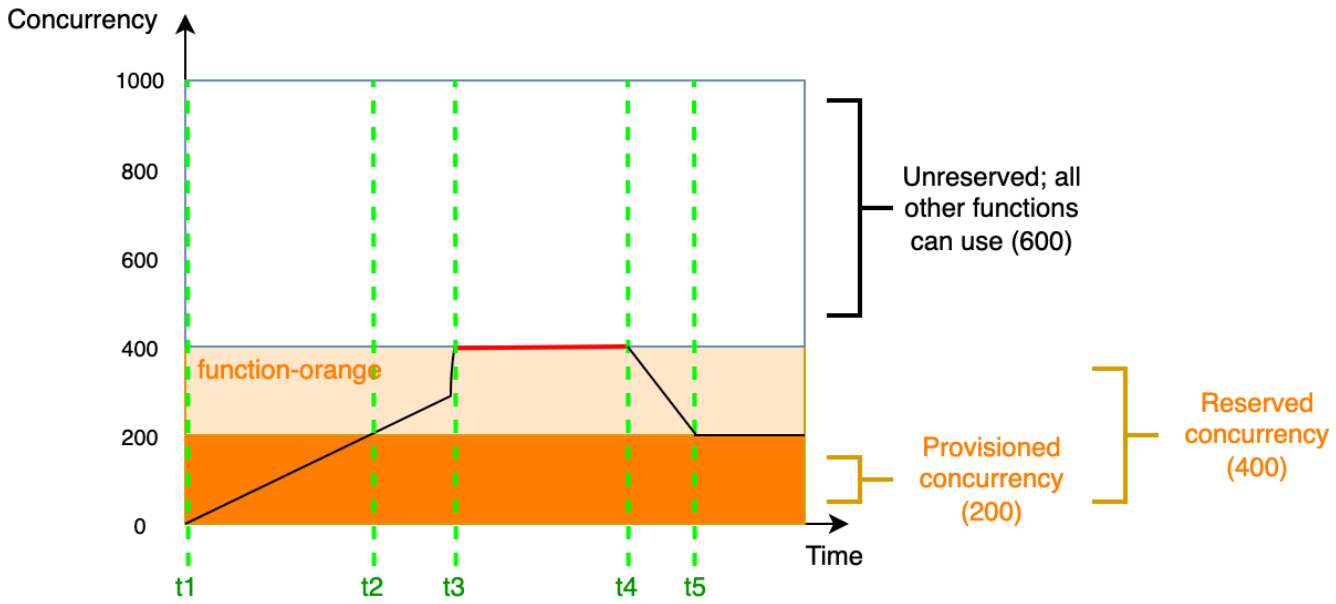
Dalam diagram ini, Anda memiliki batas konkurensi akun 1.000. Anda memutuskan untuk memberikan 400 unit konkurensi yang disediakan ke. `function-orange` Semua Fungsi Acara di akun Anda, termasuk`function-orange`, dapat menggunakan sisa 600 unit konkurensi tanpa syarat.

Diagram memiliki lima poin menarik:

- Pada`t1`, `function-orange` mulai menerima permintaan. Karena Lambda memiliki 400 instance lingkungan eksekusi yang telah diinisialisasi sebelumnya, `function-orange` siap untuk pemanggilan segera.
- Pada`t2`, `function-orange` mencapai 400 permintaan bersamaan. Akibatnya, `function-orange` kehabisan konkurensi yang disediakan. Namun, karena masih ada konkurensi tanpa syarat yang tersedia, Lambda dapat menggunakan ini untuk menangani permintaan tambahan `function-orange` (tidak ada pembatasan). Lambda harus membuat instance baru untuk melayani permintaan ini, dan Fungsi Acara Anda mungkin mengalami latensi start dingin.
- Pada`t3`, `function-orange` kembali ke 400 permintaan bersamaan setelah lonjakan lalu lintas singkat. Lambda sekali lagi dapat menangani semua permintaan tanpa latensi start dingin.

- Dit4, Fungsi Acara di akun Anda mengalami ledakan lalu lintas. Burst ini dapat berasal dari `function-orange` atau Fungsi Acara lainnya di akun Anda. Lambda menggunakan konkurensi tanpa syarat untuk menangani permintaan ini.
- Padat5, Fungsi Acara di akun Anda mencapai batas konkurensi maksimum 1.000, dan mengalami pembatasan.

Contoh sebelumnya dianggap hanya konkurensi yang disediakan. Dalam praktiknya, Anda dapat mengatur konkurensi yang disediakan dan konkurensi cadangan pada Fungsi Acara. Anda dapat melakukan ini jika Anda memiliki Fungsi Acara yang menangani beban pemanggilan yang konsisten pada hari kerja, tetapi secara rutin melihat lonjakan lalu lintas di akhir pekan. Dalam hal ini, Anda dapat menggunakan konkurensi yang disediakan untuk menetapkan jumlah dasar lingkungan untuk menangani permintaan selama hari kerja, dan menggunakan konkurensi cadangan untuk menangani lonjakan akhir pekan. Pertimbangkan diagram berikut:



Dalam diagram ini, misalkan Anda mengonfigurasi 200 unit konkurensi yang disediakan dan 400 unit konkurensi cadangan untuk. `function-orange` Karena Anda mengonfigurasi konkurensi cadangan, `function-orange` tidak dapat menggunakan salah satu dari 600 unit konkurensi tanpa syarat.

Diagram ini memiliki lima poin menarik:

- Padat1, `function-orange` mulai menerima permintaan. Karena Lambda memiliki 200 instance lingkungan eksekusi yang telah diinisialisasi sebelumnya, `function-orange` siap untuk pemanggilan segera.
- Dit2, `function-orange` menggunakan semua konkurensi yang disediakan. `function-orange`dapat terus melayani permintaan menggunakan konkurensi cadangan, tetapi permintaan ini mungkin mengalami latensi awal yang dingin.
- Padat3, `function-orange` mencapai 400 permintaan bersamaan. Akibatnya, `function-orange` menggunakan semua konkurensi yang dicadangkan. Karena `function-orange` tidak dapat menggunakan konkurensi tanpa syarat, permintaan mulai melambat.
- Padat4, `function-orange` mulai menerima lebih sedikit permintaan, dan tidak lagi throttle.
- Padat5, `function-orange` turun ke 200 permintaan bersamaan, sehingga semua permintaan kembali dapat menggunakan konkurensi yang disediakan (yaitu, tidak ada latensi start dingin).

Konkurensi cadangan dan konkurensi yang disediakan dihitung terhadap batas konkurensi akun Anda dan kuota Regional. Dengan kata lain, mengalokasikan konkurensi yang dicadangkan dan disediakan dapat memengaruhi kumpulan konkurensi yang tersedia untuk Fungsi Peristiwa lainnya. Mengkonfigurasi konkurensi yang disediakan menimbulkan biaya ke Anda. Akun AWS

> **Note**
>
> Jika jumlah konkurensi yang disediakan pada versi dan alias Fungsi Acara ditambahkan ke konkurensi cadangan Fungsi Acara, maka semua pemanggilan berjalan pada konkurensi yang disediakan. Konfigurasi ini juga memiliki efek membatasi versi Event Function (`$LATEST`) yang tidak dipublikasikan, yang mencegahnya mengeksekusi. Anda tidak dapat mengalokasikan lebih banyak konkurensi yang disediakan daripada konkurensi cadangan untuk Fungsi Peristiwa.

Untuk mengelola setelan konkurensi yang disediakan untuk Fungsi Acara Anda, lihat. Mengkonfigurasi konkurensi yang disediakan untuk Fungsi Peristiwa Untuk mengotomatiskan penskalaan konkurensi yang disediakan berdasarkan jadwal atau pemanfaatan aplikasi, lihat. Menggunakan Application Auto Scaling untuk mengotomatiskan manajemen konkurensi yang disediakan

Bagaimana Lambda mengalokasikan konkurensi yang disediakan

Konkurensi yang disediakan tidak langsung online setelah Anda mengonfigurasinya. Lambda mulai mengalokasikan konkurensi terprovisi setelah satu atau dua menit persiapan. Untuk setiap Fungsi Acara, Lambda dapat menyediakan hingga 6.000 lingkungan eksekusi setiap menit, terlepas dari itu. Wilayah AWS Ini persis sama dengan tingkat penskalaan konkurensi untuk Fungsi Acara.

Saat Anda mengirimkan permintaan untuk mengalokasikan konkurensi yang disediakan, Anda tidak dapat mengakses salah satu lingkungan tersebut hingga Lambda benar-benar selesai mengalokasikannya. Misalnya, jika Anda meminta 5.000 konkurensi yang disediakan, tidak ada permintaan yang dapat menggunakan konkurensi yang disediakan hingga Lambda benar-benar selesai mengalokasikan 5.000 lingkungan eksekusi.

Membandingkan konkurensi cadangan dan konkurensi yang disediakan

Tabel berikut merangkum dan membandingkan konkurensi yang dicadangkan dan disediakan.

Topik	Konkurensi terpesan	Konkurensi yang disediakan
Definisi	Jumlah maksimum instance lingkungan eksekusi untuk Fungsi Acara Anda.	Tetapkan jumlah instance lingkungan eksekusi yang telah disediakan sebelumnya untuk Fungsi Acara Anda.
Perilaku penyediaan	Lambda menyediakan instans baru berdasarkan permintaan.	Instans pra-ketentuan Lambda (yaitu, sebelum Fungsi Acara Anda mulai menerima permintaan).
Perilaku awal dingin	Latensi start dingin dimungkin kan, karena Lambda harus membuat instance baru sesuai permintaan.	Latensi awal dingin tidak dimungkinkan, karena Lambda tidak harus membuat instance sesuai permintaan.
Perilaku melambat	Fungsi Peristiwa dibatasi ketika batas konkurensi cadangan tercapai.	Jika konkurensi cadangan tidak disetel: Fungsi Acara menggunakan konkurens i tanpa syarat saat batas

Topik	Konkurensi terpesan	Konkurensi yang disediakan
		konkurensi yang disediakan tercapai. Jika set konkurensi cadangan: Fungsi Peristiwa dibatasi saat batas konkurensi cadangan tercapai.
Perilaku default jika tidak disetel	Fungsi Acara menggunakan konkurensi tanpa syarat yang tersedia di akun Anda.	Lambda tidak menyediak an instans apa pun terlebih dahulu. Sebaliknya, jika konkurensi cadangan tidak disetel: Fungsi Acara menggunakan konkurensi tanpa syarat yang tersedia di akun Anda. Jika set konkurensi cadangan: Fungsi Acara menggunakan konkurensi cadangan.
Harga	Tidak ada biaya tambahan.	Mengalami biaya tambahan.

Memahami konkurensi dan permintaan per detik

Seperti disebutkan di bagian sebelumnya, konkurensi berbeda dari permintaan per detik. Ini adalah perbedaan yang sangat penting untuk dibuat ketika bekerja dengan Fungsi Acara yang memiliki durasi permintaan rata-rata kurang dari 100 ms.

Di semua Fungsi Acara di akun Anda, Lambda memberlakukan batas permintaan per detik yang sama dengan 10 kali konkurensi akun Anda. Misalnya, karena batas konkurensi akun default adalah 1.000, Fungsi Acara di akun Anda dapat menangani maksimum 10.000 permintaan per detik.

Misalnya, pertimbangkan Fungsi Acara dengan durasi permintaan rata-rata 50 ms. Pada 20.000 permintaan per detik, inilah konkurensi Fungsi Acara ini:

```
Concurrency = (20,000 requests/second) * (0.05 second/request) = 1,000
```

Berdasarkan hasil ini, Anda mungkin berharap bahwa batas konkurensi akun 1.000 cukup untuk menangani beban ini. Namun, karena batas 10.000 permintaan per detik, Fungsi Acara Anda hanya dapat menangani 10.000 permintaan per detik dari total 20.000 permintaan. Fungsi Acara ini mengalami pelambatan.

Pelajarannya adalah Anda harus mempertimbangkan konkurensi dan permintaan per detik saat mengonfigurasi pengaturan konkurensi untuk Fungsi Acara Anda. Dalam hal ini, Anda perlu meminta peningkatan batas konkurensi akun menjadi 2.000, karena ini akan meningkatkan total permintaan per batas detik Anda menjadi 20.000.

> **Note**
>
> Berdasarkan batas permintaan per detik ini, tidak benar untuk mengatakan bahwa setiap lingkungan eksekusi Lambda hanya dapat menangani maksimum 10 permintaan per detik. Alih-alih mengamati beban pada setiap lingkungan eksekusi individu, Lambda hanya mempertimbangkan keseluruhan konkurensi dan permintaan keseluruhan per detik saat menghitung kuota Anda.

Uji pemahaman Anda tentang konkurensi (Fungsi Acara sub-100 ms)

Misalkan Anda memiliki Fungsi Acara yang membutuhkan, rata-rata, 20 ms untuk dijalankan. Selama beban puncak, Anda mengamati 30.000 permintaan per detik. Apa konkurensi Fungsi Acara Anda selama beban puncak?

Jawaban

Durasi Fungsi Acara rata-rata adalah 20 ms, atau 0,02 detik. Dengan menggunakan rumus konkurensi, Anda dapat memasukkan angka untuk mendapatkan konkurensi 600:

```
Concurrency = (30,000 requests/second) * (0.02 seconds/request) = 600
```

Secara default, batas konkurensi akun 1.000 tampaknya cukup untuk menangani beban ini. Namun, batas permintaan per detik 10.000 tidak cukup untuk menangani 30.000 permintaan per detik yang masuk. Untuk sepenuhnya mengakomodasi 30.000 permintaan, Anda perlu meminta peningkatan batas konkurensi akun menjadi 3.000 atau lebih tinggi.

Batas permintaan per detik berlaku untuk semua kuota di Lambda yang melibatkan konkurensi. Dengan kata lain, ini berlaku untuk Fungsi Acara sesuai permintaan sinkron, Fungsi Peristiwa yang menggunakan konkurensi yang disediakan, dan perilaku penskalaan konkurensi. Misalnya, berikut adalah beberapa skenario di mana Anda harus mempertimbangkan dengan cermat batas konkurensi dan permintaan per detik Anda:

- Fungsi Acara yang menggunakan konkurensi sesuai permintaan dapat mengalami peningkatan ledakan 500 konkurensi setiap 10 detik, atau 5.000 permintaan per detik setiap 10 detik, mana yang terjadi lebih dulu.
- Misalkan Anda memiliki Fungsi Acara yang memiliki alokasi konkurensi yang disediakan 10. Fungsi Peristiwa ini tumpah ke konkurensi sesuai permintaan setelah 10 konkurensi atau 100 permintaan per detik, mana yang terjadi lebih dulu.

Kuota konkurensi

Lambda menetapkan kuota untuk jumlah total konkurensi yang dapat Anda gunakan di semua Fungsi Peristiwa di Wilayah. Kuota ini ada pada dua tingkatan:

- Pada tingkat akun, Fungsi Acara Anda dapat memiliki hingga 1.000 unit konkurensi secara default. Untuk meningkatkan batas ini, lihat Meminta peningkatan kuota pada Panduan Pengguna Service Quotas.
- Pada tingkat fungsi, Anda dapat memesan hingga 900 unit konkurensi di semua Fungsi Acara Anda secara default. Terlepas dari total batas konkurensi akun Anda, Lambda selalu mencadangkan 100 unit konkurensi untuk Fungsi Acara Anda yang tidak secara eksplisit mencadangkan konkurensi. Misalnya, jika Anda meningkatkan batas konkurensi akun Anda menjadi 2.000, maka Anda dapat memesan hingga 1.900 unit konkurensi di tingkat fungsi.
- Pada tingkat akun dan tingkat fungsi, Lambda juga memberlakukan batas permintaan per detik sebesar 10 kali kuota konkurensi yang sesuai. Misalnya, ini berlaku untuk konkurensi tingkat akun, Fungsi Acara menggunakan konkurensi sesuai permintaan, Fungsi Acara menggunakan konkurensi yang ditentukan, dan perilaku penskalaan konkurensi. Untuk informasi selengkapnya, lihat the section called “Memahami konkurensi dan permintaan per detik”.

Untuk memeriksa kuota konkurensi tingkat akun Anda saat ini, gunakan AWS Command Line Interface (AWS CLI) untuk menjalankan perintah berikut:

```
aws lambda get-account-settings
```

Anda akan melihat output yang terlihat seperti berikut:

```
{
    "AccountLimit": {
        "TotalCodeSize": 80530636800,
        "CodeSizeUnzipped": 262144000,
        "CodeSizeZipped": 52428800,
        "ConcurrentExecutions": 1000,
        "UnreservedConcurrentExecutions": 900
    },
    "AccountUsage": {
        "TotalCodeSize": 410759889,
        "FunctionCount": 8
    }
}
```

`ConcurrentExecutions`adalah total kuota konkurensi tingkat akun Anda. `UnreservedConcurrentExecutions`adalah jumlah konkurensi cadangan yang masih dapat Anda alokasikan ke Fungsi Acara Anda.

Saat Fungsi Acara Anda menerima lebih banyak permintaan, Lambda secara otomatis meningkatkan jumlah lingkungan eksekusi untuk menangani permintaan ini hingga akun Anda mencapai kuota konkurensinya. Namun, untuk melindungi dari penskalaan berlebih sebagai respons terhadap ledakan lalu lintas yang tiba-tiba, Lambda membatasi seberapa cepat fungsi Anda dapat menskalakan. Tingkat penskalaan konkurensi ini adalah tingkat maksimum di mana Fungsi Acara di akun Anda dapat menskalakan sebagai respons terhadap peningkatan permintaan. (Artinya, seberapa cepat Lambda dapat membuat lingkungan eksekusi baru.) Tingkat penskalaan konkurensi berbeda dari batas konkurensi tingkat akun, yang merupakan jumlah total konkurensi yang tersedia untuk Fungsi Acara Anda.

Di masing-masing Wilayah AWS, dan untuk setiap Fungsi Acara, tingkat penskalaan konkurensi Anda adalah 1.000 instance lingkungan eksekusi setiap 10 detik (atau 10.000 permintaan per detik setiap 10 detik). Dengan kata lain, setiap 10 detik, Lambda dapat mengalokasikan paling banyak 1.000 instance lingkungan eksekusi tambahan, atau mengakomodasi 10.000 permintaan tambahan per detik, ke setiap fungsi Anda.

Biasanya, Anda tidak perlu khawatir tentang batasan ini. Tingkat penskalaan Lambda cukup untuk sebagian besar kasus penggunaan.

Yang penting, tingkat penskalaan konkurensi adalah batas tingkat fungsi. Ini berarti bahwa setiap Fungsi Acara di akun Anda dapat menskalakan secara independen dari Fungsi Acara lainnya.

Untuk informasi selengkapnya tentang perilaku penskalaan, lihatPerilaku penskalaan Fungsi Acara.

Mengkonfigurasi konkurensi cadangan untuk Fungsi Acara

Di Lambda, konkurensi adalah jumlah permintaan dalam penerbangan yang saat ini ditangani oleh Fungsi Acara Anda. Ada dua tipe kontrol konkurensi yang tersedia:

- Konkurensi cadangan - Ini menetapkan jumlah maksimum dan minimum instans bersamaan yang dialokasikan ke Fungsi Acara Anda. Ketika Fungsi Peristiwa telah mencadangkan konkurensi, tidak ada Fungsi Peristiwa lain yang dapat menggunakan konkurensi itu. Konkurensi cadangan berguna untuk memastikan bahwa fungsi Anda yang paling penting selalu memiliki konkurensi yang cukup untuk menangani permintaan yang masuk. Selain itu, konkurensi cadangan dapat digunakan untuk membatasi konkurensi untuk mencegah sumber daya hilir yang berlebihan, seperti koneksi database. Konkurensi cadangan bertindak sebagai batas bawah dan atas - ia menyimpan kapasitas yang ditentukan secara eksklusif untuk fungsi Anda sementara juga mencegahnya dari penskalaan melampaui batas itu. Mengkonfigurasi konkurensi cadangan untuk suatu fungsi tidak menimbulkan biaya tambahan.
- Konkurensi yang disediakan - Ini adalah jumlah lingkungan eksekusi pra-inisialisasi yang dialokasikan ke Fungsi Acara Anda. Lingkungan eksekusi ini siap merespons segera permintaan fungsi yang masuk. Konkurensi yang disediakan berguna untuk mengurangi latensi start dingin untuk fungsi dan dirancang untuk membuat fungsi tersedia dengan waktu respons milidetik dua digit. Umumnya, beban kerja interaktif paling diuntungkan dari fitur ini. Itu adalah aplikasi dengan pengguna yang memulai permintaan, seperti aplikasi web dan seluler, dan paling sensitif terhadap latensi. Beban kerja asinkron, seperti pipa pemrosesan data, seringkali kurang sensitif terhadap latensi sehingga biasanya tidak memerlukan konkurensi yang disediakan. Mengkonfigurasi konkurensi yang disediakan menimbulkan biaya tambahan untuk Anda. Akun AWS

Topik ini merinci cara mengelola dan mengonfigurasi konkurensi cadangan. Untuk gambaran konseptual dari dua jenis kontrol konkurensi ini, lihat Konkurensi cadangan dan konkurensi yang disediakan. Untuk informasi tentang mengonfigurasi konkurensi yang disediakan, lihat. the section called “Mengonfigurasi konkurensi yang tersedia”

> **Note**
>
> Fungsi Peristiwa Lambda yang ditautkan ke pemetaan sumber peristiwa Amazon MQ memiliki konkurensi maksimum default. Untuk Apache Active MQ, jumlah maksimum instans bersamaan adalah 5. Untuk Rabbit MQ, jumlah maksimum instance bersamaan adalah 1. Menyetel konkurensi cadangan atau yang disediakan untuk Fungsi Acara Anda tidak

mengubah batasan ini. Untuk meminta peningkatan konkurensi maksimum default saat menggunakan Amazon MQ, hubungi. Dukungan

Bagian-bagian

- Mengonfigurasi konkurensi terpesan
- Memperkirakan secara akurat konkurensi cadangan yang diperlukan untuk Fungsi Acara

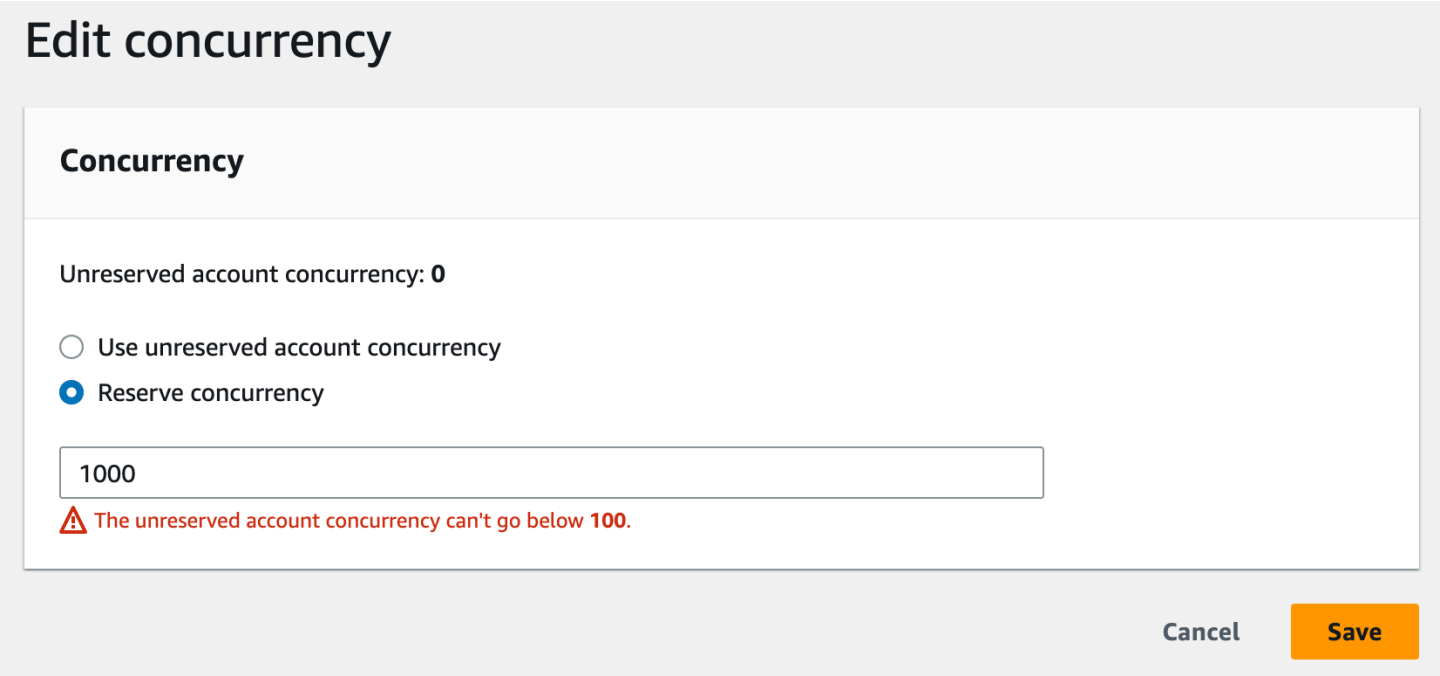
Mengonfigurasi konkurensi terpesan

Anda dapat mengonfigurasi pengaturan konkurensi cadangan untuk Fungsi Peristiwa Lambda menggunakan konsol Lambda atau API Lambda.

Untuk mencadangkan konkurensi untuk Fungsi Acara (konsol)

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih Fungsi Acara yang ingin Anda pesan konkurensi.
3. Pilih Konfigurasi, lalu pilih Konkurensi.
4. Dalam Konkurensi, pilih Edit.
5. Pilih Pesan konkurensi. Masukkan jumlah konkurensi untuk dicadangkan untuk Fungsi Acara.
6. Pilih Simpan.

Anda dapat melakukan reservasi hingga nilai konkurensi akun Unreserved dikurangi 100. 100 unit konkurensi yang tersisa adalah untuk Fungsi Acara yang tidak menggunakan konkurensi cadangan. Misalnya, jika akun Anda memiliki batas konkurensi 1.000, Anda tidak dapat memesan semua 1.000 unit konkurensi ke satu Fungsi Acara.



Reservasi konkurensi untuk Fungsi Acara berdampak pada kumpulan konkurensi yang tersedia untuk Fungsi Acara lainnya. Misalnya, jika Anda memesan 100 unit konkurensi untuk`function-a`, Fungsi Acara lain di akun Anda harus berbagi 900 unit konkurensi yang tersisa, bahkan jika `function-a` tidak menggunakan semua 100 unit konkurensi cadangan.

Untuk sengaja membatasi Fungsi Peristiwa, atur konkurensi cadangannya ke 0. Ini menghentikan Fungsi Acara Anda dari memproses peristiwa apa pun hingga Anda menghapus batasnya.

Untuk mengonfigurasi konkurensi cadangan dengan Lambda API, gunakan operasi API berikut.

- PutFunctionConcurrency
- GetFunctionConcurrency
- DeleteFunctionConcurrency

Misalnya, untuk mengonfigurasi konkurensi cadangan dengan AWS Command Line Interface (CLI), gunakan `put-function-concurrency` perintah. Perintah berikut menyimpan 100 unit konkurensi untuk Fungsi Acara bernama`my-function`:

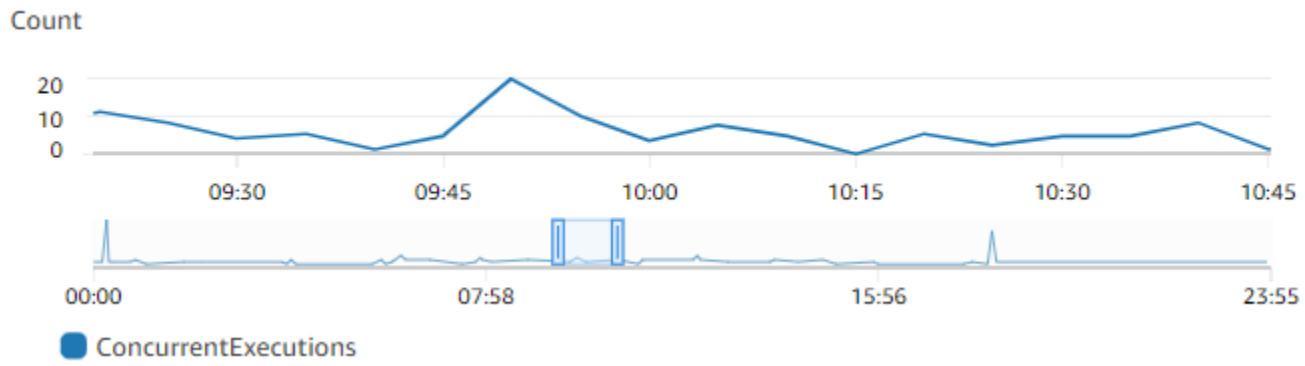
```
aws lambda put-function-concurrency --function-name my-function \
    --reserved-concurrent-executions 100
```

Anda akan melihat output yang terlihat seperti berikut:

```
{
    "ReservedConcurrentExecutions": 100
}
```

Memperkirakan secara akurat konkurensi cadangan yang diperlukan untuk Fungsi Acara

Jika Fungsi Acara Anda saat ini melayani lalu lintas, Anda dapat dengan mudah melihat metrik konkurensi menggunakan CloudWatch metrik. Secara khusus, `ConcurrentExecutions` metrik menunjukkan jumlah pemanggilan bersamaan untuk setiap Fungsi Acara di akun Anda.



Grafik sebelumnya menunjukkan bahwa Fungsi Acara ini melayani rata-rata 5 hingga 10 permintaan bersamaan pada waktu tertentu, dan memuncak pada 20 permintaan pada hari biasa. Misalkan ada banyak Fungsi Acara lainnya di akun Anda. Jika Fungsi Acara ini sangat penting untuk aplikasi Anda dan Anda tidak ingin membatalkan permintaan apa pun, gunakan angka yang lebih besar dari atau sama dengan 20 sebagai pengaturan konkurensi cadangan Anda.

Atau, ingatlah bahwa Anda juga dapat menghitung konkurensi menggunakan rumus berikut:

```
Concurrency = (average requests per second) * (average request duration in seconds)
```

Mengalikan permintaan rata-rata per detik dengan durasi permintaan rata-rata dalam hitungan detik memberi Anda perkiraan kasar tentang berapa banyak konkurensi yang perlu Anda pesan. Anda dapat memperkirakan permintaan rata-rata per detik menggunakan `Invocation` metrik, dan durasi permintaan rata-rata dalam detik menggunakan `Duration` metrik. Lihat Menggunakan CloudWatch metrik dengan Lambda untuk detail selengkapnya.

Anda juga harus terbiasa dengan kendala throughput hulu dan hilir Anda. Sementara Fungsi Peristiwa Lambda menskalakan secara mulus dengan beban, dependensi hulu dan hilir mungkin

tidak memiliki kemampuan throughput yang sama. Jika Anda perlu membatasi seberapa tinggi skala Fungsi Acara Anda, konfigurasikan konkurensi cadangan pada Fungsi Acara Anda.

Mengkonfigurasi konkurensi yang disediakan untuk Fungsi Peristiwa

Di Lambda, konkurensi adalah jumlah permintaan dalam penerbangan yang saat ini ditangani oleh Fungsi Acara Anda. Ada dua tipe kontrol konkurensi yang tersedia:

- Konkurensi cadangan - Ini menetapkan jumlah maksimum dan minimum instans bersamaan yang dialokasikan ke Fungsi Acara Anda. Ketika Fungsi Peristiwa telah mencadangkan konkurensi, tidak ada Fungsi Peristiwa lain yang dapat menggunakan konkurensi itu. Konkurensi cadangan berguna untuk memastikan bahwa fungsi Anda yang paling penting selalu memiliki konkurensi yang cukup untuk menangani permintaan yang masuk. Selain itu, konkurensi cadangan dapat digunakan untuk membatasi konkurensi untuk mencegah sumber daya hilir yang berlebihan, seperti koneksi basis data. Konkurensi cadangan bertindak sebagai batas bawah dan atas - ia menyimpan kapasitas yang ditentukan secara eksklusif untuk fungsi Anda sementara juga mencegahnya dari penskalaan melampaui batas itu. Mengkonfigurasi konkurensi cadangan untuk suatu fungsi tidak menimbulkan biaya tambahan.
- Konkurensi yang disediakan - Ini adalah jumlah lingkungan eksekusi pra-inisialisasi yang dialokasikan ke Fungsi Acara Anda. Lingkungan eksekusi ini siap merespons segera permintaan fungsi yang masuk. Konkurensi yang disediakan berguna untuk mengurangi latensi start dingin untuk fungsi dan dirancang untuk membuat fungsi tersedia dengan waktu respons milidetik dua digit. Umumnya, beban kerja interaktif paling diuntungkan dari fitur ini. Itu adalah aplikasi dengan pengguna yang memulai permintaan, seperti aplikasi web dan seluler, dan paling sensitif terhadap latensi. Beban kerja asinkron, seperti pipa pemrosesan data, seringkali kurang sensitif terhadap latensi sehingga biasanya tidak memerlukan konkurensi yang disediakan. Mengkonfigurasi konkurensi yang disediakan menimbulkan biaya tambahan untuk Anda. Akun AWS

Topik ini menjelaskan cara mengelola dan mengonfigurasi konkurensi yang disediakan. Untuk gambaran konseptual dari dua jenis kontrol konkurensi ini, lihat Konkurensi cadangan dan konkurensi yang disediakan. Untuk informasi selengkapnya tentang mengonfigurasi konkurensi cadangan, lihat. the section called “Mengonfigurasi konkurensi terpesan”

> **Note**
>
> Fungsi Peristiwa Lambda yang ditautkan ke pemetaan sumber peristiwa Amazon MQ memiliki konkurensi maksimum default. Untuk Apache Active MQ, jumlah maksimum instans bersamaan adalah 5. Untuk Rabbit MQ, jumlah maksimum instance bersamaan adalah 1. Menyetel konkurensi cadangan atau yang disediakan untuk Fungsi Acara Anda tidak

mengubah batasan ini. Untuk meminta peningkatan konkurensi maksimum default saat menggunakan Amazon MQ, hubungi. Dukungan

Bagian-bagian

- Mengonfigurasi konkurensi yang tersedia
- Secara akurat memperkirakan konkurensi yang disediakan yang diperlukan untuk Fungsi Peristiwa
- Mengoptimalkan kode Fungsi Peristiwa saat menggunakan konkurensi yang disediakan
- Menggunakan variabel lingkungan untuk melihat dan mengontrol perilaku konkurensi yang disediakan
- Memahami perilaku pencatatan dan penagihan dengan konkurensi yang disediakan
- Menggunakan Application Auto Scaling untuk mengotomatiskan manajemen konkurensi yang disediakan

Mengonfigurasi konkurensi yang tersedia

Anda dapat mengonfigurasi pengaturan konkurensi yang disediakan untuk Fungsi Peristiwa Lambda menggunakan konsol Lambda atau API Lambda.

Untuk mengalokasikan konkurensi yang disediakan untuk Fungsi Acara (konsol)

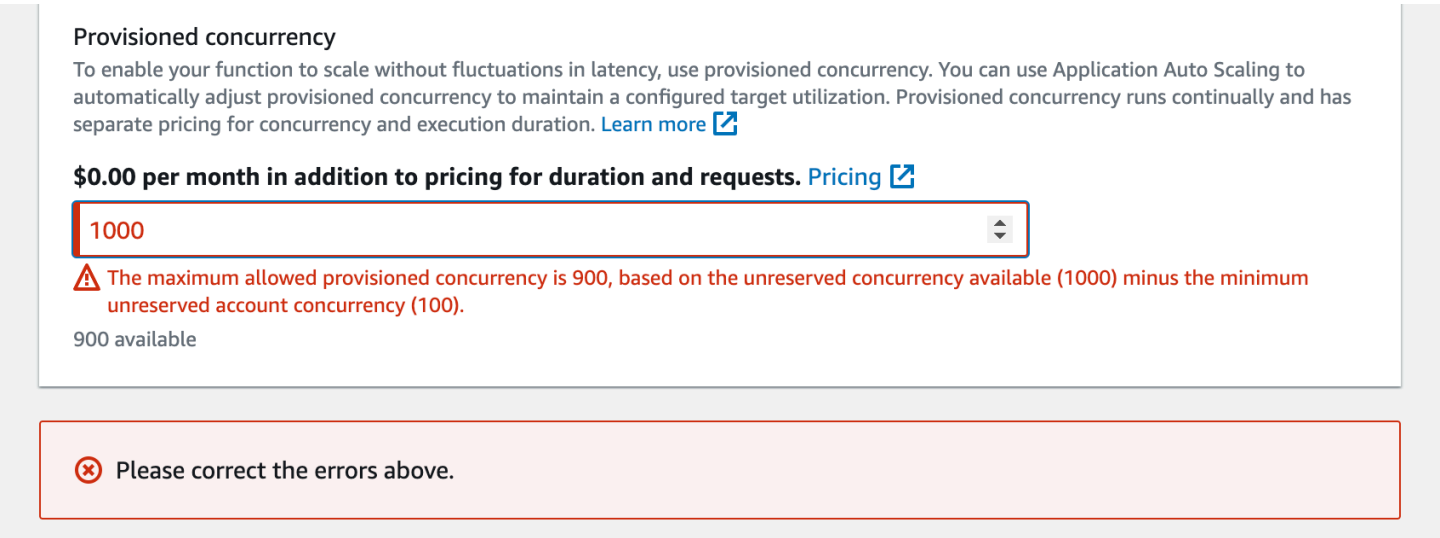
1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih Fungsi Acara yang ingin Anda alokasikan konkurensi yang disediakan.
3. Pilih Konfigurasi, lalu pilih Konkurensi.
4. Dalam Konfigurasi konkurensi terprovisi, pilih Tambah konfigurasi.
5. Pilih jenis qualifier, dan alias atau versi.

> **Note**
>
> Anda tidak dapat menggunakan konkurensi yang disediakan dengan versi $LATEST dari Fungsi Acara apa pun.
> Jika Event Function Anda memiliki sumber peristiwa, pastikan bahwa sumber peristiwa menunjuk ke alias atau versi Event Function yang benar. Jika tidak, Fungsi Acara Anda tidak akan menggunakan lingkungan konkurensi yang disediakan.

6. Masukkan nomor di bawah Konkurensi yang disediakan.

7. Pilih Simpan.

Anda dapat mengonfigurasi hingga konkurensi akun Unreserved di akun Anda, minus 100. 100 unit konkurensi yang tersisa adalah untuk Fungsi Acara yang tidak menggunakan konkurensi cadangan. Misalnya, jika akun Anda memiliki batas konkurensi 1.000, dan Anda belum menetapkan konkurensi cadangan atau ketentuan apa pun ke Fungsi Acara lainnya, Anda dapat mengonfigurasi maksimum 900 unit konkurensi yang disediakan untuk satu Fungsi Acara.



Mengkonfigurasi konkurensi yang disediakan untuk Fungsi Peristiwa berdampak pada kumpulan konkurensi yang tersedia untuk Fungsi Peristiwa lainnya. Misalnya, jika Anda mengonfigurasi 100 unit konkurensi yang disediakan untuk`function-a`, Fungsi Acara lainnya di akun Anda harus membagikan 900 unit konkurensi yang tersisa. Ini benar bahkan jika `function-a` tidak menggunakan semua 100 unit.

Dimungkinkan untuk mengalokasikan konkurensi cadangan dan konkurensi yang disediakan untuk Fungsi Peristiwa yang sama. Dalam kasus seperti itu, konkurensi yang disediakan tidak dapat melebihi konkurensi yang dicadangkan.

Batasan ini meluas ke versi Fungsi Acara. Konkurensi maksimum yang disediakan yang dapat Anda tetapkan ke versi Fungsi Peristiwa tertentu adalah konkurensi cadangan Fungsi Acara dikurangi konkurensi yang disediakan pada versi Fungsi Peristiwa lainnya.

Untuk mengonfigurasi konkurensi yang disediakan dengan API Lambda, gunakan operasi API berikut.

- PutProvisionedConcurrencyConfig

- GetProvisionedConcurrencyConfig
- ListProvisionedConcurrencyConfigs
- DeleteProvisionedConcurrencyConfig

Misalnya, untuk mengkonfigurasi konkurensi yang disediakan dengan (AWS Command Line Interface CLI), gunakan perintah. `put-provisioned-concurrency-config` Perintah berikut mengalokasikan 100 unit konkurensi yang disediakan untuk `BLUE` alias Fungsi Peristiwa bernama: `my-function`

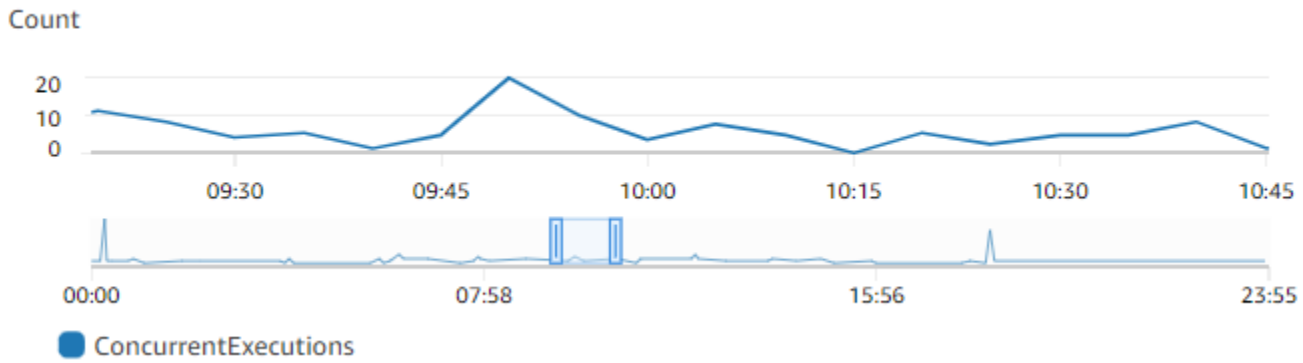
```
aws lambda put-provisioned-concurrency-config --function-name my-function \
  --qualifier BLUE \
  --provisioned-concurrent-executions 100
```

Anda akan melihat output yang terlihat seperti berikut:

```
{
  "Requested ProvisionedConcurrentExecutions": 100,
  "Allocated ProvisionedConcurrentExecutions": 0,
  "Status": "IN_PROGRESS",
  "LastModified": "2023-01-21T11:30:00+0000"
}
```

Secara akurat memperkirakan konkurensi yang disediakan yang diperlukan untuk Fungsi Peristiwa

Anda dapat melihat metrik konkurensi Fungsi Acara yang aktif menggunakan CloudWatch metrik. Secara khusus, `ConcurrentExecutions` metrik menunjukkan jumlah pemanggilan bersamaan untuk Fungsi Peristiwa di akun Anda.



Grafik sebelumnya menunjukkan bahwa Fungsi Acara ini melayani rata-rata 5 hingga 10 permintaan bersamaan pada waktu tertentu, dan memuncak pada 20 permintaan. Misalkan ada banyak Fungsi Acara lainnya di akun Anda. Jika Fungsi Peristiwa ini sangat penting untuk aplikasi Anda dan Anda memerlukan respons latensi rendah pada setiap pemanggilan, konfigurasikan setidaknya 20 unit konkurensi yang disediakan.

Ingatlah bahwa Anda juga dapat menghitung konkurensi menggunakan rumus berikut:

```
Concurrency = (average requests per second) * (average request duration in seconds)
```

Untuk memperkirakan berapa banyak konkurensi yang Anda butuhkan, kalikan permintaan rata-rata per detik dengan durasi permintaan rata-rata dalam hitungan detik. Anda dapat memperkirakan permintaan rata-rata per detik menggunakan `Invocation` metrik, dan durasi permintaan rata-rata dalam detik menggunakan `Duration` metrik.

Saat mengonfigurasi konkurensi yang disediakan, Lambda menyarankan untuk menambahkan buffer 10% di atas jumlah konkurensi yang biasanya dibutuhkan Fungsi Acara Anda. Misalnya, jika Fungsi Acara Anda biasanya memuncak pada 200 permintaan bersamaan, setel konkurensi yang disediakan ke 220 (200 permintaan bersamaan + 10% = 220 konkurensi yang disediakan).

Mengoptimalkan kode Fungsi Peristiwa saat menggunakan konkurensi yang disediakan

Jika Anda menggunakan konkurensi yang disediakan, pertimbangkan untuk merestrukturisasi kode Fungsi Acara Anda untuk mengoptimalkan latensi rendah. Untuk Fungsi Peristiwa yang menggunakan konkurensi yang disediakan, Lambda menjalankan kode inisialisasi apa pun, seperti memuat pustaka dan membuat instance klien, selama waktu alokasi. Oleh karena itu, disarankan untuk memindahkan sebanyak mungkin inisialisasi di luar penangan Fungsi Acara utama untuk menghindari dampak latensi selama pemanggilan Fungsi Peristiwa yang sebenarnya. Sebaliknya, menginisialisasi pustaka atau membuat instance klien dalam kode pengendali utama Anda berarti Fungsi Acara Anda harus menjalankan ini setiap kali dipanggil (ini terjadi terlepas dari apakah Anda menggunakan konkurensi yang disediakan).

Untuk pemanggilan sesuai permintaan, Lambda mungkin perlu menjalankan ulang kode inisialisasi Anda setiap kali Fungsi Acara Anda mengalami awal yang dingin. Untuk Fungsi Acara seperti itu, Anda dapat memilih untuk menunda inisialisasi kemampuan tertentu hingga Fungsi Acara Anda membutuhkannya. Misalnya, pertimbangkan alur kontrol berikut untuk handler Lambda:

```
def handler(event, context):
```

```
    ...
    if ( some_condition ):
        // Initialize CLIENT_A to perform a task
    else:
        // Do nothing
```

Pada contoh sebelumnya, alih-alih menginisialisasi `CLIENT_A` di luar handler utama, pengembang menginisialisasinya dalam pernyataan. `if` Dengan melakukan ini, Lambda menjalankan kode ini hanya jika `some_condition` terpenuhi. Jika Anda menginisialisasi `CLIENT_A` di luar penangan utama, Lambda menjalankan kode itu di setiap awal yang dingin. Ini dapat meningkatkan latensi keseluruhan.

Anda dapat mengukur start dingin saat Lambda meningkatkan skala dengan menambahkan X-Ray pemantauan ke Fungsi Acara Anda. Fungsi Peristiwa yang menggunakan konkurensi yang disediakan tidak menunjukkan perilaku start dingin karena lingkungan eksekusi disiapkan sebelum pemanggilan. Namun, konkurensi yang disediakan harus diterapkan ke versi atau alias tertentu dari Fungsi Acara, bukan versi $LATEST. Dalam kasus di mana Anda terus melihat perilaku start dingin, pastikan Anda menjalankan versi alias dengan konkurensi yang disediakan yang dikonfigurasi.

Menggunakan variabel lingkungan untuk melihat dan mengontrol perilaku konkurensi yang disediakan

Fungsi Acara Anda dapat menggunakan semua konkurensi yang disediakan. Lambda menggunakan instans sesuai permintaan untuk menangani kelebihan lalu lintas. Untuk menentukan jenis inisialisasi Lambda yang digunakan untuk lingkungan tertentu, periksa nilai variabel lingkungan. `AWS_LAMBDA_INITIALIZATION_TYPE`

Variabel ini memiliki dua nilai yang mungkin: `provisioned-concurrency` atau`on-demand`. Nilai `AWS_LAMBDA_INITIALIZATION_TYPE` tidak dapat diubah dan tetap konstan sepanjang masa hidup lingkungan. Untuk memeriksa nilai variabel lingkungan dalam kode Event Function Anda, lihatMengambil variabel lingkungan Lambda.

Jika Anda menggunakan runtime .NET 8, Anda dapat mengonfigurasi variabel `AWS_LAMBDA_DOTNET_PREJIT` lingkungan untuk meningkatkan latensi untuk Fungsi Acara, meskipun mereka tidak menggunakan konkurensi yang disediakan. Runtime.NET menggunakan kompilasi dan inisialisasi malas untuk setiap pustaka yang dipanggil kode Anda untuk pertama kalinya. Akibatnya, pemanggilan pertama dari Fungsi Peristiwa Lambda mungkin memakan waktu lebih lama daripada yang berikutnya. Untuk mengurangi ini, Anda dapat memilih salah satu dari tiga nilai untuk: `AWS_LAMBDA_DOTNET_PREJIT`

- `ProvisionedConcurrency`: Lambda melakukan kompilasi JIT sebelumnya untuk semua lingkungan menggunakan konkurensi yang disediakan. Ini adalah nilai default.
- `Always`: Lambda melakukan kompilasi JIT sebelumnya untuk setiap lingkungan, bahkan jika Fungsi Acara tidak menggunakan konkurensi yang disediakan.
- `Never`: Lambda menonaktifkan kompilasi JIT sebelumnya untuk semua lingkungan.

Memahami perilaku pencatatan dan penagihan dengan konkurensi yang disediakan

Untuk lingkungan konkurensi yang disediakan, kode inisialisasi Fungsi Acara Anda berjalan selama alokasi, dan secara berkala saat Lambda mendaur ulang instance lingkungan Anda. Lambda menagih Anda untuk inisialisasi meskipun instance lingkungan tidak pernah memproses permintaan. Konkurensi yang disediakan berjalan terus menerus dan menimbulkan penagihan terpisah dari biaya inisialisasi dan pemanggilan. Untuk detail selengkapnya, lihat AWS Lambda Harga.

Saat Anda mengonfigurasi Fungsi Peristiwa Lambda dengan konkurensi yang disediakan, Lambda melakukan pra-inisialisasi lingkungan eksekusi tersebut sehingga tersedia sebelum permintaan pemanggilan. Lambda mencatat bidang Durasi Init dari Fungsi Peristiwa dalam peristiwa log Platform-initReport dalam format logging JSON setiap kali lingkungan diinisialisasi. Untuk melihat peristiwa log ini, konfigurasikan level log JSON Anda ke setidaknya`INFO`. Anda juga dapat menggunakan API Telemetri untuk menggunakan peristiwa platform tempat bidang Durasi Init dilaporkan.

Menggunakan Application Auto Scaling untuk mengotomatiskan manajemen konkurensi yang disediakan

Anda dapat menggunakan Application Auto Scaling untuk mengelola konkurensi yang disediakan sesuai jadwal atau berdasarkan pemanfaatan. Jika Fungsi Acara Anda menerima pola lalu lintas yang dapat diprediksi, gunakan penskalaan terjadwal. Jika Anda ingin Fungsi Acara mempertahankan persentase pemanfaatan tertentu, gunakan kebijakan penskalaan pelacakan target.

> **Note**
>
> Jika Anda menggunakan Application Auto Scaling untuk mengelola konkurensi yang disediakan oleh Fungsi Acara, pastikan Anda mengonfigurasi nilai konkurensi awal yang disediakan terlebih dahulu. Jika Fungsi Acara Anda tidak memiliki nilai konkurensi awal yang

disediakan, Application Auto Scaling mungkin tidak menangani penskalaan Fungsi Peristiwa dengan benar.

Penskalaan terjadwal

Dengan Application Auto Scaling, Anda dapat mengatur jadwal penskalaan Anda sendiri sesuai dengan perubahan beban yang dapat diprediksi. Untuk informasi dan contoh selengkapnya, lihat Penskalaan terjadwal untuk Application Auto Scaling di Panduan Pengguna Application Auto Scaling, AWS Lambda dan Scheduling Provisioned Concurrency untuk penggunaan puncak berulang di Blog Komputasi. AWS

Pelacakan Target

Dengan pelacakan target, Application Auto Scaling membuat dan mengelola serangkaian CloudWatch alarm berdasarkan cara Anda menentukan kebijakan penskalaan Anda. Ketika alarm ini diaktifkan, Application Auto Scaling secara otomatis menyesuaikan jumlah lingkungan yang dialokasikan menggunakan konkurensi yang disediakan. Gunakan pelacakan target untuk aplikasi yang tidak memiliki pola lalu lintas yang dapat diprediksi.

Untuk menskalakan konkurensi yang disediakan menggunakan pelacakan target, gunakan operasi Application Auto Scaling API `RegisterScalableTarget` dan Application Auto `PutScalingPolicy` Scaling. Misalnya, jika Anda menggunakan AWS Command Line Interface (CLI), ikuti langkah-langkah berikut:

1. Daftarkan alias Event Function sebagai target penskalaan. Contoh berikut mendaftarkan alias BLUE dari Fungsi Peristiwa bernama: `my-function`

```
aws application-autoscaling register-scalable-target --service-namespace lambda \
    --resource-id function:my-function:BLUE --min-capacity 1 --max-capacity 100 \
    --scalable-dimension lambda:function:ProvisionedConcurrency
```

2. Terapkan kebijakan penskalaan ke target. Contoh berikut mengonfigurasi Application Auto Scaling untuk menyesuaikan konfigurasi konkurensi yang disediakan untuk alias agar pemanfaatan tetap mendekati 70 persen, tetapi Anda dapat menerapkan nilai apa pun antara 10% dan 90%.

```
aws application-autoscaling put-scaling-policy \
    --service-namespace lambda \
    --scalable-dimension lambda:function:ProvisionedConcurrency \
```

```
    --resource-id function:my-function:BLUE \
    --policy-name my-policy \
    --policy-type TargetTrackingScaling \
    --target-tracking-scaling-policy-configuration '{ "TargetValue":
 0.7, "PredefinedMetricSpecification": { "PredefinedMetricType":
 "LambdaProvisionedConcurrencyUtilization" }}'
```

Anda akan melihat output seperti ini:

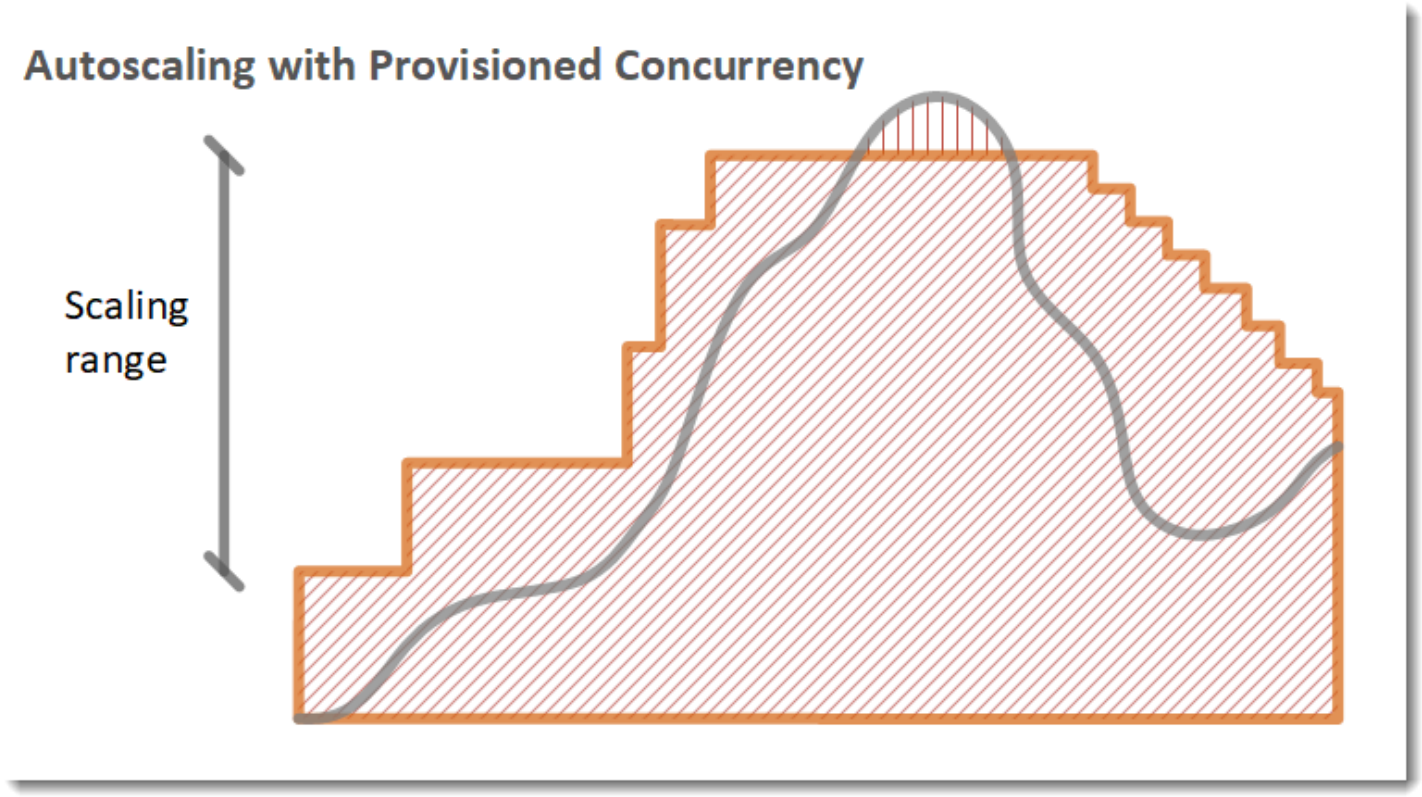
```
{
    "PolicyARN": "arn:aws:autoscaling:us-
east-2:123456789012:scalingPolicy:12266dbb-1524-xmpl-a64e-9a0a34b996fa:resource/lambda/
function:my-function:BLUE:policyName/my-policy",
    "Alarms": [
        {
            "AlarmName": "TargetTracking-function:my-function:BLUE-AlarmHigh-aed0e274-
xmpl-40fe-8cba-2e78f000c0a7",
            "AlarmARN": "arn:aws:cloudwatch:us-
east-2:123456789012:alarm:TargetTracking-function:my-function:BLUE-AlarmHigh-aed0e274-
xmpl-40fe-8cba-2e78f000c0a7"
        },
        {
            "AlarmName": "TargetTracking-function:my-function:BLUE-AlarmLow-7e1a928e-
xmpl-4d2b-8c01-782321bc6f66",
            "AlarmARN": "arn:aws:cloudwatch:us-
east-2:123456789012:alarm:TargetTracking-function:my-function:BLUE-AlarmLow-7e1a928e-
xmpl-4d2b-8c01-782321bc6f66"
        }
    ]
}
```

Application Auto Scaling membuat dua alarm masuk. CloudWatch Alarm pertama terpicu ketika pemanfaatan konkurensi yang disediakan secara konsisten melebihi 70%. Jika hal ini terjadi, Application Auto Scaling mengalokasikan lebih banyak konkurensi terprovisi untuk mengurangi penggunaan. Alarm kedua terpicu ketika pemanfaatan secara konsisten kurang dari 63% (90 persen dari target 70%). Jika hal ini terjadi, Application Auto Scaling mengurangi konkurensi terprovisi alias.

> **Note**
>
> Lambda memancarkan `ProvisionedConcurrencyUtilization` metrik hanya jika Fungsi Acara Anda aktif dan menerima permintaan. Selama periode tidak aktif, tidak ada metrik

yang dipancarkan, dan alarm auto-scaling Anda memasuki status. `INSUFFICIENT_DATA` Akibatnya, Application Auto Scaling tidak akan dapat menyesuaikan konkurensi yang disediakan Event Function Anda. Ini dapat menyebabkan penagihan yang tidak terduga.

Dalam contoh berikut, Fungsi Peristiwa menskalakan antara jumlah minimum dan maksimum konkurensi yang disediakan berdasarkan pemanfaatan.



Legenda

- Fungsi Acara
- Permintaan terbuka
- Konkurensi terprovisi
- Konkurensi standar

Contoh

Ketika jumlah permintaan terbuka meningkat, Application Auto Scaling meningkatkan konkurensi yang disediakan dalam langkah-langkah besar hingga mencapai maksimum yang dikonfigurasi. Setelah ini, Fungsi Acara dapat terus menskalakan pada konkurensi standar tanpa syarat jika Anda belum mencapai batas konkurensi akun Anda. Saat pemanfaatan turun dan tetap rendah, Application Auto Scaling mengurangi konkurensi yang disediakan dalam langkah-langkah periodik yang lebih kecil.

Kedua alarm Application Auto Scaling menggunakan statistik rata-rata secara default. Fungsi yang mengalami semburan lalu lintas cepat mungkin tidak memicu alarm ini. Misalnya, Fungsi Acara Lambda Anda dijalankan dengan cepat (yaitu 20-100 ms) dan lalu lintas Anda datang dalam ledakan cepat. Dalam hal ini, jumlah permintaan melebihi konkurensi yang disediakan yang dialokasikan selama burst.

Namun, Application Auto Scaling membutuhkan beban burst untuk bertahan setidaknya selama 3 menit untuk menyediakan lingkungan tambahan. Selain itu, kedua CloudWatch alarm memerlukan 3 titik data yang mencapai rata-rata target untuk mengaktifkan kebijakan penskalaan otomatis. Jika Fungsi Acara Anda mengalami ledakan lalu lintas yang cepat, menggunakan statistik Maksimum alih-alih statistik Rata-rata dapat lebih efektif dalam menskalakan konkurensi yang disediakan untuk meminimalkan start dingin.

Untuk informasi selengkapnya tentang kebijakan penskalaan pelacakan target, lihat Kebijakan penskalaan pelacakan target untuk Application Auto Scaling.

Perilaku penskalaan Fungsi Acara

Saat Fungsi Acara Anda menerima lebih banyak permintaan, Lambda secara otomatis meningkatkan jumlah lingkungan eksekusi untuk menangani permintaan ini hingga akun Anda mencapai kuota konkurensinya. Namun, untuk melindungi dari penskalaan berlebih sebagai respons terhadap ledakan lalu lintas yang tiba-tiba, Lambda membatasi seberapa cepat fungsi Anda dapat menskalakan. Tingkat penskalaan konkurensi ini adalah tingkat maksimum di mana Fungsi Acara di akun Anda dapat menskalakan sebagai respons terhadap peningkatan permintaan. (Artinya, seberapa cepat Lambda dapat membuat lingkungan eksekusi baru.) Tingkat penskalaan konkurensi berbeda dari batas konkurensi tingkat akun, yang merupakan jumlah total konkurensi yang tersedia untuk Fungsi Acara Anda.

Tingkat penskalaan konkurensi

Di masing-masing Wilayah AWS, dan untuk setiap Fungsi Acara, tingkat penskalaan konkurensi Anda adalah 1.000 instance lingkungan eksekusi setiap 10 detik (atau 10.000 permintaan per detik setiap 10 detik). Dengan kata lain, setiap 10 detik, Lambda dapat mengalokasikan paling banyak 1.000 instance lingkungan eksekusi tambahan, atau mengakomodasi 10.000 permintaan tambahan per detik, ke setiap fungsi Anda.

Biasanya, Anda tidak perlu khawatir tentang batasan ini. Tingkat penskalaan Lambda cukup untuk sebagian besar kasus penggunaan.

Yang penting, tingkat penskalaan konkurensi adalah batas tingkat fungsi. Ini berarti bahwa setiap Fungsi Acara di akun Anda dapat menskalakan secara independen dari Fungsi Acara lainnya.

> **Note**
>
> Dalam praktiknya, Lambda melakukan upaya terbaik untuk mengisi ulang tingkat penskalaan konkurensi Anda secara terus menerus dari waktu ke waktu, daripada dalam satu isi ulang 1.000 unit setiap 10 detik.

Lambda tidak memperoleh bagian yang tidak terpakai dari tingkat penskalaan konkurensi Anda. Ini berarti bahwa setiap saat, tingkat penskalaan Anda selalu maksimum 1.000 unit konkurensi. Misalnya, jika Anda tidak menggunakan 1.000 unit konkurensi yang tersedia dalam interval 10 detik, Anda tidak akan memperoleh 1.000 unit tambahan dalam interval 10 detik berikutnya. Tingkat penskalaan konkurensi Anda masih 1.000 dalam interval 10 detik berikutnya.

Selama Fungsi Acara Anda terus menerima peningkatan jumlah permintaan, maka Lambda menskalakan pada tingkat tercepat yang tersedia untuk Anda, hingga batas konkurensi akun Anda. Anda dapat membatasi jumlah konkurensi yang dapat digunakan oleh Fungsi Peristiwa individual dengan mengonfigurasi konkurensi cadangan. Jika permintaan masuk lebih cepat dari skala Fungsi Acara Anda, atau jika Fungsi Acara Anda berada pada konkurensi maksimum, maka permintaan tambahan gagal dengan kesalahan pelambatan (kode status 429).

Memantau konkurensi

Lambda memancarkan metrik CloudWatch Amazon untuk membantu Anda memantau konkurensi untuk Fungsi Peristiwa Lambda Anda. Topik ini menjelaskan metrik ini dan cara menafsirkannya.

Bagian-bagian

- Metrik konkurensi umum
- Metrik konkurensi terprovisi
- Bekerja dengan ClaimedAccountConcurrencymetrik

Metrik konkurensi umum

Gunakan metrik berikut untuk memantau konkurensi Fungsi Peristiwa Lambda Anda. Granularitas untuk setiap metrik adalah 1 menit.

- `ConcurrentExecutions`— Jumlah pemanggilan bersamaan aktif pada titik waktu tertentu. Lambda memancarkan metrik ini untuk semua Fungsi Acara, versi, dan alias. Untuk Fungsi Peristiwa apa pun di konsol Lambda, Lambda menampilkan grafik **`ConcurrentExecutions`** secara native di tab Pemantauan, di bawah Metrik. Lihat metrik ini menggunakan MAX.
- `UnreservedConcurrentExecutions`— Jumlah pemanggilan bersamaan aktif yang menggunakan konkurensi tanpa syarat. Lambda memancarkan metrik ini di semua Fungsi Peristiwa di suatu wilayah. Lihat metrik ini menggunakan MAX.
- `ClaimedAccountConcurrency`— Jumlah konkurensi yang tidak tersedia untuk pemanggilan sesuai permintaan. `ClaimedAccountConcurrency`sama dengan `UnreservedConcurrentExecutions` ditambah jumlah konkurensi yang dialokasikan (yaitu total konkurensi cadangan ditambah total konkurensi yang disediakan). Jika `ClaimedAccountConcurrency` melebihi batas konkurensi akun Anda, Anda dapat meminta batas konkurensi akun yang lebih tinggi.

  Lihat metrik ini menggunakan MAX. Untuk informasi selengkapnya, lihat Bekerja dengan `ClaimedAccountConcurrency`metrik.

Metrik konkurensi terprovisi

Gunakan metrik berikut untuk memantau Fungsi Peristiwa Lambda menggunakan konkurensi yang disediakan. Granularitas untuk setiap metrik adalah 1 menit.

- `ProvisionedConcurrentExecutions`— Jumlah instance lingkungan eksekusi yang secara aktif memproses pemanggilan pada konkurensi yang disediakan. Lambda memancarkan metrik ini untuk setiap versi Fungsi Acara dan alias dengan konkurensi yang disediakan yang dikonfigurasi. Lihat metrik ini menggunakan MAX.

`ProvisionedConcurrentExecutions`tidak sama dengan jumlah total konkurensi yang disediakan yang Anda alokasikan. Misalnya, Anda mengalokasikan 100 unit konkurensi yang disediakan ke versi Fungsi Acara. Selama menit tertentu, jika paling banyak 50 dari 100 lingkungan eksekusi tersebut menangani pemanggilan secara bersamaan, maka nilai MAX (`ProvisionedConcurrentExecutions`) adalah 50.

- `ProvisionedConcurrencyInvocations`— Berapa kali Lambda memanggil kode Fungsi Acara Anda menggunakan konkurensi yang disediakan. Lambda memancarkan metrik ini untuk setiap versi Fungsi Acara dan alias dengan konkurensi yang disediakan yang dikonfigurasi. Lihat metrik ini menggunakan SUM.

`ProvisionedConcurrencyInvocations`berbeda dari `ProvisionedConcurrentExecutions` yang `ProvisionedConcurrencyInvocations` menghitung jumlah total pemanggilan, sementara `ProvisionedConcurrentExecutions` menghitung jumlah lingkungan aktif. Untuk memahami perbedaan ini, pertimbangkan skenario berikut:



Dalam contoh ini, misalkan Anda menerima 1 doa per menit, dan setiap doa membutuhkan waktu 2 menit untuk menyelesaikannya. Setiap bar horisontal oranye mewakili satu permintaan. Misalkan Anda mengalokasikan 10 unit konkurensi yang disediakan ke Fungsi Peristiwa ini, sehingga setiap permintaan berjalan pada konkurensi yang disediakan.

Di antara menit 0 dan 1, `Request 1` masuk. Pada menit 1, nilai untuk MAX (`ProvisionedConcurrentExecutions`) adalah 1, karena paling banyak 1 lingkungan eksekusi aktif selama beberapa menit terakhir. Nilai untuk SUM (`ProvisionedConcurrencyInvocations`) juga 1, karena 1 permintaan baru masuk selama beberapa menit terakhir.

Di antara menit 1 dan 2, `Request 2` masuk, dan `Request 1` terus berjalan. Pada menit 2, nilai untuk MAX (`ProvisionedConcurrentExecutions`) adalah 2, karena paling banyak 2 lingkungan eksekusi aktif selama beberapa menit terakhir. Namun, nilai untuk SUM (`ProvisionedConcurrencyInvocations`) adalah 1, karena hanya 1 permintaan baru yang masuk selama beberapa menit terakhir. Perilaku metrik ini berlanjut hingga akhir contoh.

- `ProvisionedConcurrencySpilloverInvocations`— Berapa kali Lambda memanggil Fungsi Acara Anda pada konkurensi standar (reserved atau unreserved) saat semua konkurensi yang disediakan sedang digunakan. Lambda memancarkan metrik ini untuk setiap versi Fungsi Acara dan alias dengan konkurensi yang disediakan yang dikonfigurasi. Lihat metrik ini menggunakan SUM. Nilai `ProvisionedConcurrencyInvocations` + `ProvisionedConcurrencySpilloverInvocations` harus sama dengan jumlah total pemanggilan Fungsi Peristiwa (yaitu `Invocations` metrik).

  `ProvisionedConcurrencyUtilization`— Persentase konkurensi yang disediakan yang digunakan (yaitu nilai `ProvisionedConcurrentExecutions` dibagi dengan jumlah total konkurensi yang disediakan yang dialokasikan). Lambda memancarkan metrik ini untuk setiap versi Fungsi Acara dan alias dengan konkurensi yang disediakan yang dikonfigurasi. Lihat metrik ini menggunakan MAX.

Misalnya, Anda menyediakan 100 unit konkurensi yang disediakan ke versi Fungsi Acara. Selama menit tertentu, jika paling banyak 60 dari 100 lingkungan eksekusi tersebut menangani pemanggilan secara bersamaan, maka nilai MAX (`ProvisionedConcurrentExecutions`) adalah 60, dan nilai MAX (`ProvisionedConcurrencyUtilization`) adalah 0,6.

Nilai tinggi untuk `ProvisionedConcurrencySpilloverInvocations` mungkin menunjukkan bahwa Anda perlu mengalokasikan konkurensi tambahan yang disediakan untuk Fungsi Acara Anda. Atau, Anda dapat mengonfigurasi Application Auto Scaling untuk menangani penskalaan otomatis konkurensi yang disediakan berdasarkan ambang batas yang telah ditentukan sebelumnya.

Sebaliknya, nilai rendah secara konsisten untuk `ProvisionedConcurrencyUtilization` mungkin menunjukkan bahwa Anda mengalokasikan konkurensi yang disediakan secara berlebihan untuk Fungsi Acara Anda.

Bekerja dengan **`ClaimedAccountConcurrency`**metrik

Lambda menggunakan `ClaimedAccountConcurrency` metrik untuk menentukan berapa banyak konkurensi akun Anda tersedia untuk pemanggilan sesuai permintaan. Lambda menghitung `ClaimedAccountConcurrency` menggunakan rumus berikut:

```
ClaimedAccountConcurrency = UnreservedConcurrentExecutions + (allocated concurrency)
```

`UnreservedConcurrentExecutions`adalah jumlah pemanggilan bersamaan aktif yang menggunakan konkurensi tanpa syarat. Konkurensi yang dialokasikan adalah jumlah dari dua bagian berikut (diganti sebagai “konkurensi cadangan” dan RC PC sebagai “konkurensi yang disediakan”):

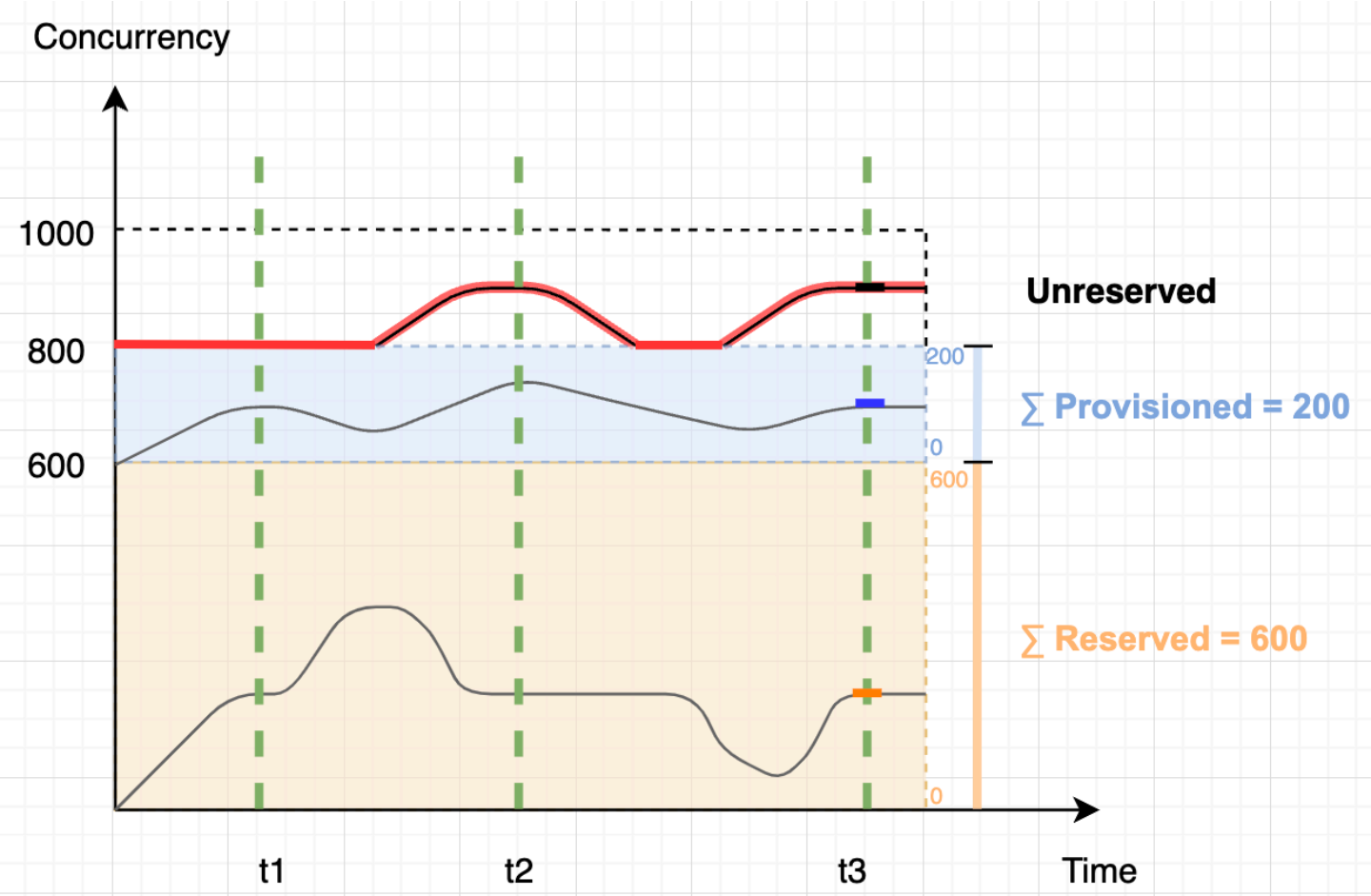
- Total RC di semua Fungsi Acara di Wilayah.
- Total PC di semua Fungsi Peristiwa di Wilayah yang menggunakanPC, tidak termasuk Fungsi Peristiwa yang digunakanRC.

> **Note**
>
> Anda tidak dapat mengalokasikan PC lebih dari RC untuk Fungsi Acara. Dengan demikian, Fungsi Peristiwa RC selalu lebih besar dari atau sama dengan fungsinyaPC. Untuk menghitung kontribusi konkurensi yang dialokasikan untuk Fungsi Acara tersebut dengan keduanya PC dan, RC Lambda hanya mempertimbangkanRC, yang merupakan maksimum dari keduanya.

Lambda menggunakan `ClaimedAccountConcurrency` metrik, bukan`ConcurrentExecutions`, untuk menentukan berapa banyak konkurensi yang tersedia untuk pemanggilan sesuai permintaan. Meskipun `ConcurrentExecutions` metrik berguna untuk melacak jumlah pemanggilan bersamaan yang aktif, metrik tidak selalu mencerminkan ketersediaan konkurensi Anda yang sebenarnya. Ini karena Lambda juga mempertimbangkan konkurensi cadangan dan konkurensi yang disediakan untuk menentukan ketersediaan.

Untuk mengilustrasikan`ClaimedAccountConcurrency`, pertimbangkan skenario di mana Anda mengonfigurasi banyak konkurensi cadangan dan konkurensi yang disediakan di seluruh Fungsi Acara Anda yang sebagian besar tidak digunakan. Dalam contoh berikut, asumsikan bahwa batas konkurensi akun Anda adalah 1.000, dan Anda memiliki dua Fungsi Acara utama di akun Anda: `function-orange` dan`function-blue`. Anda mengalokasikan 600 unit konkurensi cadangan

untuk. `function-orange` Anda mengalokasikan 200 unit konkurensi yang disediakan untuk. `function-blue` Misalkan seiring waktu, Anda menerapkan Fungsi Peristiwa tambahan dan mengamati pola lalu lintas berikut:



Pada diagram sebelumnya, garis hitam menunjukkan penggunaan konkurensi aktual dari waktu ke waktu, dan garis merah menunjukkan nilai dari waktu `ClaimedAccountConcurrency` ke waktu. Sepanjang skenario `ClaimedAccountConcurrency` ini, minimal 800, meskipun pemanfaatan konkurensi aktual rendah di seluruh Fungsi Acara Anda. Ini karena Anda mengalokasikan total 800 unit konkurensi untuk `function-orange` dan. `function-blue` Dari perspektif Lambda, Anda telah "mengklaim" konkurensi ini untuk digunakan, sehingga Anda secara efektif hanya memiliki 200 unit konkurensi yang tersisa untuk Fungsi Acara lainnya.

Untuk skenario ini, konkurensi yang dialokasikan adalah 800 dalam rumus. `ClaimedAccountConcurrency` Kita kemudian dapat memperoleh nilai `ClaimedAccountConcurrency` pada berbagai titik dalam diagram:

- Padat`1`, `ClaimedAccountConcurrency` adalah 800 (800 + `0UnreservedConcurrentExecutions`).

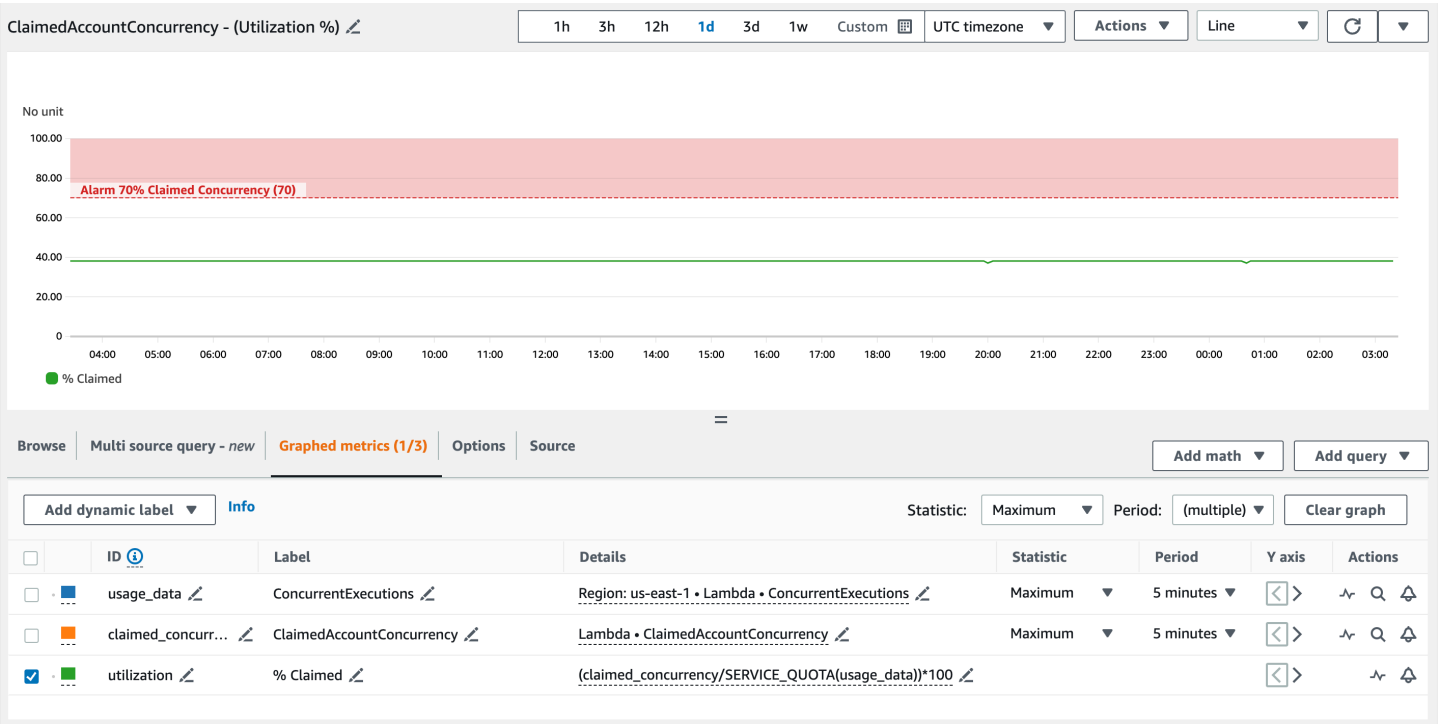
- Pada `t2`, `ClaimedAccountConcurrency` adalah 900 (800 + `UnreservedConcurrentExecutions`).
- Pada `t3`, `ClaimedAccountConcurrency` lagi 900 (800 + `UnreservedConcurrentExecutions`).

Menyiapkan **`ClaimedAccountConcurrency`**metrik di CloudWatch

Lambda memancarkan metrik di. `ClaimedAccountConcurrency` CloudWatch Gunakan metrik ini bersama dengan nilai `SERVICE_QUOTA(ConcurrentExecutions)` untuk mendapatkan persentase pemanfaatan konkurensi di akun Anda, seperti yang ditunjukkan dalam rumus berikut:

```
Utilization = (ClaimedAccountConcurrency/SERVICE_QUOTA(ConcurrentExecutions)) * 100%
```

Screenshot berikut menggambarkan bagaimana Anda dapat membuat grafik rumus ini. CloudWatch `claim_utilization`Garis hijau mewakili pemanfaatan konkurensi dalam akun ini, yaitu sekitar 40%:



Tangkapan layar sebelumnya juga menyertakan CloudWatch alarm yang masuk ke `ALARM` status ketika pemanfaatan konkurensi melebihi 70%. Anda dapat menggunakan `ClaimedAccountConcurrency` metrik bersama dengan alarm serupa untuk secara proaktif menentukan kapan Anda mungkin perlu meminta batas konkurensi akun yang lebih tinggi.

Pemantauan, debugging, dan pemecahan masalah Fungsi Peristiwa Lambda

AWS Lambda terintegrasi dengan yang lain Layanan AWS untuk membantu Anda memantau dan memecahkan masalah Fungsi Acara Lambda Anda. Lambda secara otomatis memonitor Fungsi Acara atas nama Anda dan melaporkan metrik melalui Amazon. CloudWatch Untuk membantu Anda memantau kode ketika dijalankan, Lambda secara otomatis melacak jumlah permintaan, durasi invokasi per permintaan, dan jumlah permintaan yang menghasilkan kesalahan.

Anda dapat menggunakan yang lain Layanan AWS untuk memecahkan masalah Fungsi Acara Lambda Anda. Bagian ini menjelaskan cara menggunakannya Layanan AWS untuk memantau, melacak, men-debug, dan memecahkan masalah Fungsi dan aplikasi Peristiwa Lambda Anda. Untuk detail tentang logging Fungsi Peristiwa dan kesalahan di setiap runtime, lihat bagian runtime individual. Topik meliputi pemantauan aplikasi, Sinyal Aplikasi, dan pencatatan panggilan API Lambda dengan. CloudTrail

Bagian-bagian

- Harga
- Menggunakan CloudWatch metrik dengan Lambda
- Bekerja dengan log Fungsi Peristiwa Lambda
- Pencatatan log AWS Lambda Panggilan API menggunakan AWS CloudTrail
- Visualisasikan pemanggilan Fungsi Acara Lambda menggunakan AWS X-Ray
- Pantau kinerja Fungsi Acara dengan Amazon CloudWatch Lambda Insights
- Memantau aplikasi Lambda
- Pantau kinerja aplikasi dengan Sinyal CloudWatch Aplikasi Amazon
- Debug dari jarak jauh Fungsi Acara Lambda dengan Kode Visual Studio

Harga

CloudWatch memiliki tingkat bebas abadi. Di luar ambang batas tingkat gratis, CloudWatch biaya untuk metrik, dasbor, alarm, log, dan wawasan. Untuk informasi selengkapnya, lihat CloudWatch harga Amazon.

Menggunakan CloudWatch metrik dengan Lambda

Saat Fungsi AWS Lambda Acara selesai memproses peristiwa, Lambda secara otomatis mengirimkan metrik tentang pemanggilan ke Amazon. CloudWatch Anda tidak perlu memberikan izin tambahan apa pun ke peran eksekusi untuk menerima metrik Fungsi Acara, dan tidak ada biaya tambahan untuk metrik ini.

Ada banyak jenis metrik yang terkait dengan Fungsi Peristiwa Lambda. Ini termasuk metrik pemanggilan, metrik kinerja, metrik konkurensi, metrik pemanggilan asinkron, dan metrik pemetaan sumber peristiwa. Untuk informasi selengkapnya, lihat the section called “Jenis metrik”.

Di CloudWatch konsol, Anda dapat melihat metrik ini dan membuat grafik dan dasbor dengannya. Anda juga dapat mengatur alarm untuk merespons perubahan dalam pemanfaatan, kinerja, atau tingkat kesalahan. Lambda mengirimkan data metrik ke CloudWatch dalam interval 1 menit.

Untuk wawasan lebih cepat tentang Fungsi Peristiwa Lambda Anda, Anda dapat membuat metrik kustom resolusi tinggi. Biaya berlaku untuk metrik dan CloudWatch alarm khusus. Untuk informasi selengkapnya, lihat CloudWatch Harga Amazon.

Melihat metrik untuk Fungsi Acara Lambda

Gunakan CloudWatch konsol untuk melihat metrik untuk Fungsi Peristiwa Lambda Anda. Di konsol, Anda dapat memfilter dan mengurutkan metrik fungsi berdasarkan nama fungsi, alias, versi, atau pemetaan sumber peristiwa UUID.

Untuk melihat metrik di konsol CloudWatch

1. Buka halaman Metrik (`AWS/Lambda`namespace) konsol. CloudWatch
2. Pada tab Browse, di bawah Metrik, pilih salah satu dimensi berikut:

   - Dengan Nama Fungsi (`FunctionName`) - Lihat metrik agregat untuk semua versi dan alias Fungsi Peristiwa.
   - By Resource (`Resource`) - Lihat metrik untuk versi atau alias Fungsi Peristiwa.
   - Menurut Versi yang Dieksekusi (`ExecutedVersion`) – Lihat metrik untuk kombinasi alias dan versi. Gunakan `ExecutedVersion` dimensi untuk membandingkan tingkat kesalahan untuk dua versi Fungsi Peristiwa yang keduanya merupakan target alias tertimbang.
   - Berdasarkan Pemetaan Sumber Peristiwa UUID (`EventSourceMappingUUID`) — Lihat metrik untuk pemetaan sumber peristiwa.

- Di Seluruh Semua Fungsi (tidak ada) - Lihat metrik agregat untuk semua Fungsi Peristiwa saat ini. Wilayah AWS

3. Pilih metrik. Metrik akan secara otomatis muncul di grafik visual, serta di bawah tab Metrik grafik.

Secara default, grafik menggunakan statistik `Sum` untuk semua metrik. Untuk memilih statistik yang berbeda dan menyesuaikan grafik, gunakan opsi pada tab Metrik bergrafik.

> **Note**
>
> Stempel waktu pada metrik mencerminkan kapan Fungsi Peristiwa dipanggil. Tergantung pada durasi pemanggilan, ini bisa beberapa menit sebelum metrik dipancarkan. Misalnya, jika Fungsi Acara Anda memiliki batas waktu 10 menit, lihat lebih dari 10 menit sebelumnya untuk metrik yang akurat.

Untuk informasi selengkapnya CloudWatch, lihat Panduan CloudWatch Pengguna Amazon.

Jenis metrik untuk Fungsi Peristiwa Lambda

Bagian ini menjelaskan jenis metrik Lambda yang tersedia di konsol. CloudWatch

Topik

- Metrik invokasi
- Metrik penyebaran
- Metrik kinerja
- Metrik konkurensi
- Metrik pemanggilan asinkron
- Metrik pemetaan sumber acara

Metrik invokasi

Metrik pemanggilan adalah indikator biner dari hasil pemanggilan Fungsi Peristiwa Lambda. Lihat metrik ini dengan `Sum` statistik. Misalnya, jika Fungsi Peristiwa mengembalikan kesalahan, maka Lambda mengirimkan `Errors` metrik dengan nilai 1. Untuk mendapatkan hitungan jumlah kesalahan Fungsi Peristiwa yang terjadi setiap menit, lihat `Errors` metrik dengan jangka waktu 1 menit. `Sum`

- `Invocations`— Berapa kali kode Fungsi Acara Anda dipanggil, termasuk pemanggilan dan pemanggilan yang berhasil yang menghasilkan kesalahan fungsi. Pemanggilan tidak direkam jika permintaan pemanggilan dibatasi atau menghasilkan kesalahan pemanggilan. Nilai `Invocations` sama dengan jumlah permintaan yang ditagih.
- `Errors` – Jumlah invokasi yang mengakibatkan kesalahan fungsi. Kesalahan fungsi mencakup pengecualian yang dilemparkan kode Anda dan pengecualian yang dilemparkan oleh runtime Lambda. Runtime mengembalikan kesalahan untuk masalah seperti waktu habis dan kesalahan konfigurasi. Untuk menghitung tingkat kesalahan, bagi nilai `Errors` dengan nilai `Invocations`. Perhatikan bahwa stempel waktu pada metrik kesalahan mencerminkan kapan Fungsi Peristiwa dipanggil, bukan saat kesalahan terjadi.
- `DeadLetterErrors`— Untuk pemanggilan asinkron, berapa kali Lambda mencoba mengirim acara ke antrian huruf mati (DLQ) tetapi gagal. Dead-letter kesalahan dapat terjadi karena salah mengatur sumber daya atau batas ukuran.
- `DestinationDeliveryFailures`— Untuk pemanggilan asinkron dan pemetaan sumber peristiwa yang didukung, berapa kali Lambda mencoba mengirim acara ke tujuan tetapi gagal. Untuk pemetaan sumber peristiwa, Lambda mendukung tujuan untuk sumber aliran (DynamoDB dan Kinesis). Kesalahan pengiriman dapat terjadi karena kesalahan izin, sumber daya yang tidak dikonfigurasi dengan benar, atau batas ukuran. Kesalahan juga dapat terjadi jika tujuan yang telah Anda konfigurasikan adalah tipe yang tidak didukung seperti antrian FIFO Amazon SQS atau topik FIFO Amazon SNS.
- `Throttles` – Jumlah permintaan invokasi yang ditrotel. Ketika semua instance fungsi memproses permintaan dan tidak ada konkurensi yang tersedia untuk ditingkatkan, Lambda menolak permintaan tambahan dengan kesalahan. `TooManyRequestsException` Permintaan yang dibatasi dan kesalahan pemanggilan lainnya tidak dihitung sebagai salah satu atau. `Invocations` `Errors`

> **Note**
>
> Dengan Instans Terkelola Lambda, Lambda menyediakan metrik throttle granular yang mengidentifikasi batasan spesifik yang menyebabkan throttle. Ketika throttle terjadi pada lingkungan eksekusi, tepat salah satu sub-metrik berikut dipancarkan dengan nilai 1, sedangkan tiga sisanya dipancarkan dengan nilai 0. `Throttles`Metrik selalu dipancarkan bersama sub-metrik ini.
>
> - `CPUThrottles`— Pemanggilan terhambat karena kelelahan CPU pada lingkungan eksekusi.

  - `MemoryThrottles`— Pemanggilan terhambat karena kelelahan memori pada lingkungan eksekusi.
  - `DiskThrottles`— Pemanggilan terhambat karena kelelahan disk pada lingkungan eksekusi.
  - `ConcurrencyThrottles`— Pemanggilan dibatasi ketika batas konkurensi lingkungan eksekusi tercapai.

- `OversizedRecordCount`— Untuk sumber acara Amazon DocumentDB, jumlah peristiwa yang diterima Fungsi Acara Anda dari aliran perubahan Anda yang berukuran lebih dari 6 MB. Lambda menjatuhkan pesan dan memancarkan metrik ini.
- `ProvisionedConcurrencyInvocations`— Berapa kali kode Fungsi Acara Anda dipanggil menggunakan konkurensi yang disediakan.
- `ProvisionedConcurrencySpilloverInvocations`— Berapa kali kode Fungsi Acara Anda dipanggil menggunakan konkurensi standar saat semua konkurensi yang disediakan sedang digunakan.
- `RecursiveInvocationsDropped`— Berapa kali Lambda menghentikan pemanggilan Fungsi Acara Anda karena telah mendeteksi bahwa Fungsi Peristiwa Anda adalah bagian dari loop rekursif tak terbatas. Deteksi loop rekursif memantau berapa kali Fungsi Peristiwa dipanggil sebagai bagian dari rantai permintaan dengan melacak metadata yang ditambahkan oleh SDK yang didukung. AWS Secara default, jika Fungsi Acara Anda dipanggil sebagai bagian dari rantai permintaan sekitar 16 kali, Lambda akan menghentikan pemanggilan berikutnya. Jika Anda menonaktifkan deteksi loop rekursif, metrik ini tidak dipancarkan. Untuk informasi selengkapnya tentang fitur ini, lihat Menggunakan Deteksi Loop Rekursif Lambda untuk Mencegah Loop Tak Terbatas.

Metrik penyebaran

Metrik penerapan memberikan informasi tentang peristiwa penerapan Fungsi Peristiwa Lambda dan proses validasi terkait.

- `SignatureValidationErrors`— Berapa kali penyebaran paket kode terjadi dengan kegagalan validasi tanda tangan saat kebijakan konfigurasi penandatanganan kode disetel ke. `Warn` Metrik ini dipancarkan saat pemeriksaan kedaluwarsa, ketidakcocokan, atau pencabutan gagal tetapi penerapan masih diizinkan karena pengaturan kebijakan. `Warn` Untuk informasi selengkapnya tentang penandatanganan kode, lihat Menggunakan penandatanganan kode untuk memverifikasi integritas kode dengan Lambda.

Metrik kinerja

Metrik kinerja memberikan detail kinerja tentang satu pemanggilan Fungsi Peristiwa. Misalnya, `Duration` metrik menunjukkan jumlah waktu dalam milidetik yang dihabiskan Fungsi Peristiwa Anda untuk memproses suatu peristiwa. Untuk mengetahui seberapa cepat Fungsi Acara memproses peristiwa, lihat metrik ini dengan `Average` atau `Max` statistik.

- `Duration`— Jumlah waktu yang dihabiskan kode Fungsi Acara Anda untuk memproses suatu peristiwa. Durasi tagihan untuk pemanggilan adalah nilai `Duration` dibulatkan ke milidetik terdekat. `Duration`tidak termasuk waktu mulai dingin.
- `PostRuntimeExtensionsDuration`— Jumlah kumulatif waktu yang dihabiskan runtime menjalankan kode untuk ekstensi setelah kode Fungsi Acara selesai.
- `IteratorAge`— Untuk sumber acara DynamoDB, Kinesis, dan Amazon DocumentDB, usia rekaman terakhir dalam acara dalam milidetik. Metrik ini mengukur waktu antara saat aliran menerima rekaman dan saat pemetaan sumber peristiwa mengirimkan peristiwa ke Fungsi Peristiwa.
- `OffsetLag`— Untuk sumber acara Apache Kafka dan Amazon Managed Streaming untuk Apache Kafka (Amazon MSK) yang dikelola sendiri, perbedaan offset antara catatan terakhir yang ditulis ke topik dan catatan terakhir yang diproses oleh grup konsumen Fungsi Acara Anda. Meskipun topik Kafka dapat memiliki beberapa partisi, metrik ini mengukur lag offset pada tingkat topik.

`Duration`juga mendukung statistik persentil (p). Gunakan persentil untuk mengecualikan nilai outlier yang miring dan statistik. `Average` `Maximum` Misalnya, `p95` statistik menunjukkan durasi maksimum 95 persen pemanggilan, tidak termasuk 5 persen paling lambat. Untuk informasi selengkapnya, lihat Persentil di CloudWatch Panduan Pengguna Amazon.

Metrik konkurensi

Lambda melaporkan metrik konkurensi sebagai jumlah agregat dari jumlah instance yang memproses peristiwa di seluruh Fungsi Peristiwa, versi, alias, atau. Wilayah AWS Untuk melihat seberapa dekat Anda dengan mencapai batas konkurensi, lihat metrik ini dengan statistik. `Max`

- `ConcurrentExecutions`— Jumlah instance Fungsi Peristiwa yang memproses peristiwa. Jika nomor ini mencapai kuota eksekusi bersamaan untuk Wilayah, atau batas konkurensi cadangan pada Fungsi Acara, maka Lambda membatasi permintaan pemanggilan tambahan.
- `ProvisionedConcurrentExecutions`— Jumlah instance Fungsi Peristiwa yang memproses peristiwa menggunakan konkurensi yang disediakan. Untuk setiap invokasi alias atau versi dengan

konkurensi tersedia, Lambda mengeluarkan jumlah saat ini. Jika Fungsi Acara Anda tidak aktif atau tidak menerima permintaan, Lambda tidak memancarkan metrik ini.

- `ProvisionedConcurrencyUtilization`— Untuk versi atau alias, nilai `ProvisionedConcurrentExecutions` dibagi dengan jumlah total konkurensi yang disediakan dikonfigurasi. Misalnya, jika Anda mengonfigurasi konkurensi 10 yang disediakan untuk Fungsi Acara Anda, dan 7, maka Anda `ProvisionedConcurrentExecutions` `ProvisionedConcurrencyUtilization` adalah 0,7.

  Jika Fungsi Acara Anda tidak aktif atau tidak menerima permintaan, Lambda tidak memancarkan metrik ini karena didasarkan pada. `ProvisionedConcurrentExecutions` Ingatlah hal ini jika Anda menggunakan `ProvisionedConcurrencyUtilization` sebagai dasar untuk CloudWatch alarm.
- `UnreservedConcurrentExecutions`— Untuk Wilayah, jumlah peristiwa yang Fungsi Acara tanpa konkurensi cadangan diproses.
- `ClaimedAccountConcurrency`— Untuk Wilayah, jumlah konkurensi yang tidak tersedia untuk pemanggilan sesuai permintaan. `ClaimedAccountConcurrency`sama dengan `UnreservedConcurrentExecutions` ditambah jumlah konkurensi yang dialokasikan (yaitu total konkurensi cadangan ditambah total konkurensi yang disediakan). Untuk informasi selengkapnya, lihat Bekerja dengan `ClaimedAccountConcurrency`metrik.

Metrik pemanggilan asinkron

Metrik pemanggilan asinkron memberikan detail tentang pemanggilan asinkron dari sumber acara dan pemanggilan langsung. Anda dapat mengatur ambang batas dan alarm untuk memberi tahu Anda tentang perubahan tertentu. Misalnya, ketika ada peningkatan yang tidak diinginkan dalam jumlah acara yang diantri untuk pemrosesan (). `AsyncEventsReceived` Atau, ketika suatu acara telah menunggu lama untuk diproses (`AsyncEventAge`).

- `AsyncEventsReceived`— Jumlah acara yang berhasil diantri Lambda untuk diproses. Metrik ini memberikan wawasan tentang jumlah peristiwa yang diterima Fungsi Peristiwa. Pantau metrik ini dan setel alarm untuk ambang batas untuk memeriksa masalah. Misalnya, untuk mendeteksi sejumlah peristiwa yang tidak diinginkan yang dikirim ke Lambda, dan untuk dengan cepat mendiagnosis masalah yang dihasilkan dari pemicu atau konfigurasi fungsi yang salah. Ketidakcocokan antara `AsyncEventsReceived` dan `Invocations` dapat menunjukkan perbedaan dalam pemrosesan, peristiwa yang dijatuhkan, atau potensi backlog antrian.

- `AsyncEventAge`— Waktu antara saat Lambda berhasil mengantri acara dan saat Fungsi Acara dipanggil. Nilai metrik ini meningkat ketika peristiwa sedang dicoba ulang karena kegagalan pemanggilan atau pelambatan. Pantau metrik ini dan atur alarm untuk ambang batas pada statistik yang berbeda saat terjadi penumpukan antrian. Untuk memecahkan masalah peningkatan metrik ini, lihat metrik untuk mengidentifikasi kesalahan fungsi dan `Errors` metrik untuk mengidentifikasi masalah konkurensi. `Throttles`
- `AsyncEventsDropped`— Jumlah peristiwa yang dijatuhkan tanpa berhasil mengeksekusi Fungsi Acara. Jika Anda mengonfigurasi antrian surat mati (DLQ) atau `OnFailure` tujuan, maka acara dikirim ke sana sebelum dijatuhkan. Acara dijatuhkan karena berbagai alasan. Misalnya, peristiwa dapat melebihi usia peristiwa maksimum atau menghabiskan upaya percobaan ulang maksimum, atau konkurensi cadangan dapat disetel ke 0. Untuk memecahkan masalah mengapa peristiwa dijatuhkan, lihat `Errors` metrik untuk mengidentifikasi kesalahan fungsi dan `Throttles` metrik untuk mengidentifikasi masalah konkurensi.

Metrik pemetaan sumber acara

Metrik pemetaan sumber peristiwa memberikan wawasan tentang perilaku pemrosesan pemetaan sumber peristiwa Anda.

Saat ini, metrik pemetaan sumber peristiwa tersedia untuk Amazon SQS, Kinesis, DynamoDB, Amazon MSK, dan sumber acara Apache Kafka yang dikelola sendiri.

Untuk pemetaan sumber peristiwa dengan konfigurasi metrik, Anda juga dapat memeriksa semua metrik terkait ESM di tab Monitor dari halaman Konsol Lambda > Sumber daya tambahan> pemetaan sumber peristiwa sekarang.

Untuk mengaktifkan metrik atau pemetaan sumber peristiwa (konsol)

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih Fungsi Acara yang ingin Anda aktifkan metrik.
3. Pilih Konfigurasi, lalu pilih Pemicu.
4. Pilih pemetaan sumber peristiwa yang ingin Anda aktifkan metrik, lalu pilih Edit.
5. Di bawah Konfigurasi pemetaan sumber peristiwa, pilih Aktifkan metrik atau pilih dari daftar dropdown Metrik.
6. Pilih Simpan.

Atau, Anda dapat mengaktifkan metrik untuk pemetaan sumber peristiwa Anda secara terprogram menggunakan objek di EventSourceMappingMetricsConfigAnda. EventSourceMappingConfiguration Misalnya, perintah UpdateEventSourceMappingCLI berikut memungkinkan metrik untuk pemetaan sumber peristiwa:

```
aws lambda update-event-source-mapping \
    --uuid a1b2c3d4-5678-90ab-cdef-EXAMPLE11111 \
    --metrics-config Metrics=EventCount
```

Ada 3 metrik goups:`EventCount`, `ErrorCount` dan`KafkaMetrics`, dan setiap grup memiliki multi metrik. Tidak setiap metrik tersedia untuk setiap sumber acara. Tabel berikut merangkum metrik yang didukung untuk setiap jenis sumber peristiwa.

Anda harus ikut serta dalam grup metrik untuk menerima metrik terkait metrik. misalnya diatur EventCount dalam konfigurasi metrik untuk memiliki: (`PolledEventCount`,,,,, `FilteredOutEventCount` `InvokedEventCount`, `FailedInvokeEventCount` dan). `DroppedEventCount` `OnFailureDestinationDeliveredEventCount` `DeletedEventCount`

Metrik pemetaan sumber peristiwa	Grup metrik	Amazon SQS	Aliran Kinesis dan DynamoDB	Amazon MSK dan Apache Kafka yang dikelola sendiri
`PolledEventCount`	`EventCount`	Ya	Ya	Ya
`FilteredOutEventCount`	`EventCount`	Ya	Ya	Ya
`InvokedEventCount`	`EventCount`	Ya	Ya	Ya
`FailedInvokeEventCount`	`EventCount`	Ya	Ya	Ya

Metrik pemetaan sumber peristiwa	Grup metrik	Amazon SQS	Aliran Kinesis dan DynamoDB	Amazon MSK dan Apache Kafka yang dikelola sendiri
DroppedEventCount	EventCount	Tidak	Ya	Ya
OnFailureDestinationDeliveredEventCount	EventCount	Tidak	Ya	Ya
DeletedEventCount	EventCount	Ya	Tidak	Tidak
CommittedEventCount	EventCount	Tidak	Tidak	Ya
PollingErrorCount	ErrorCount	Tidak	Tidak	Ya
InvokeErrorCount	ErrorCount	Tidak	Tidak	Ya
OnFailureDestinationDeliveryErrorCount	ErrorCount	Tidak	Tidak	Ya
SchemaRegistryErrorCount	ErrorCount	Tidak	Tidak	Ya
CommitErrorCount	ErrorCount	Tidak	Tidak	Ya

Metrik pemetaan sumber peristiwa	Grup metrik	Amazon SQS	Aliran Kinesis dan DynamoDB	Amazon MSK dan Apache Kafka yang dikelola sendiri
`MaxOffsetLag`	`KafkaMetrics`	Tidak	Tidak	Ya
`SumOffsetLag`	`KafkaMetrics`	Tidak	Tidak	Ya

Selain itu, jika pemetaan sumber peristiwa Anda dalam mode yang disediakan, Lambda menyediakan metrik berikut:

- `ProvisionedPollers`— Untuk pemetaan sumber acara dalam mode yang disediakan, jumlah poller acara yang aktif berjalan. Lihat metrik ini menggunakan `MAX` matematika.
- (Amazon MSK dan sumber acara Apache Kafka yang dikelola sendiri saja) `EventPollerUnit` - Untuk pemetaan sumber acara dalam mode yang disediakan, jumlah unit poller acara yang aktif berjalan. Lihat metrik ini menggunakan `SUM` matematika.
- (Amazon MSK dan sumber acara Apache Kafka yang dikelola sendiri) `EventPollerThroughputInBytes` - Untuk pemetaan sumber acara dalam mode yang disediakan, total ukuran catatan poller acara yang disurvei dari sumber acara. Ini dapat memberi tahu Anda throughput polling saat ini. Lihat metrik ini menggunakan `SUM` matematika.

Berikut ini detail lebih lanjut tentang masing-masing metrik:

- `PolledEventCount`— Jumlah acara yang berhasil dibaca Lambda dari sumber acara. Jika Lambda melakukan polling untuk acara tetapi menerima polling kosong (tidak ada catatan baru), Lambda mengeluarkan nilai 0 untuk metrik ini. Gunakan metrik ini untuk mendeteksi apakah pemetaan sumber peristiwa Anda melakukan polling dengan benar untuk peristiwa baru.
- `FilteredOutEventCount`— Untuk pemetaan sumber peristiwa dengan kriteria filter, jumlah peristiwa yang disaring oleh kriteria filter tersebut. Gunakan metrik ini untuk mendeteksi apakah pemetaan sumber peristiwa Anda memfilter peristiwa dengan benar. Untuk peristiwa yang cocok dengan kriteria filter, Lambda memancarkan metrik 0.

- `InvokedEventCount`— Jumlah acara yang memanggil Fungsi Acara Lambda Anda. Gunakan metrik ini untuk memverifikasi bahwa peristiwa menjalankan Fungsi Acara Anda dengan benar. Jika suatu peristiwa menghasilkan kesalahan fungsi atau pelambatan, `InvokedEventCount` mungkin dihitung beberapa kali untuk acara polling yang sama karena percobaan ulang otomatis.

  > **Warning**
  >
  > Pemetaan sumber peristiwa Lambda memproses setiap peristiwa setidaknya sekali, dan pemrosesan duplikat catatan dapat terjadi. Karena itu, peristiwa dapat dihitung beberapa kali dalam metrik yang melibatkan jumlah peristiwa.

- `FailedInvokeEventCount`— Jumlah peristiwa yang Lambda coba panggil Fungsi Acara Anda, tetapi gagal. Pemanggilan dapat gagal karena alasan seperti masalah konfigurasi jaringan, izin yang salah, atau Fungsi Peristiwa Lambda yang dihapus, versi, atau alias. Jika pemetaan sumber peristiwa Anda mengaktifkan respons batch sebagian, `FailedInvokeEventCount` sertakan peristiwa apa pun dengan respons yang tidak kosong`BatchItemFailures`.

  > **Note**
  >
  > Stempel waktu untuk `FailedInvokeEventCount` metrik mewakili akhir dari pemanggilan Fungsi Peristiwa. Perilaku ini berbeda dari metrik kesalahan pemanggilan Lambda lainnya, yang diberi cap waktu pada awal pemanggilan Fungsi Peristiwa.

- `DroppedEventCount`— Jumlah peristiwa yang Lambda turun karena kedaluwarsa atau coba lagi kelelahan. Secara khusus, ini adalah jumlah catatan yang melebihi nilai yang dikonfigurasi untuk `MaximumRecordAgeInSeconds` atau`MaximumRetryAttempts`. Yang penting, ini tidak termasuk jumlah catatan yang kedaluwarsa karena melebihi pengaturan retensi sumber acara Anda. Peristiwa yang dijatuhkan juga mengecualikan peristiwa yang Anda kirim ke tujuan yang gagal. Gunakan metrik ini untuk mendeteksi tumpukan peristiwa yang meningkat.
- `OnFailureDestinationDeliveredEventCount`— Untuk pemetaan sumber peristiwa dengan tujuan yang gagal dikonfigurasi, jumlah peristiwa yang dikirim ke tujuan tersebut. Gunakan metrik ini untuk memantau kesalahan fungsi yang terkait dengan pemanggilan dari sumber acara ini. Jika pengiriman ke tujuan gagal, Lambda menangani metrik sebagai berikut:
  - Lambda tidak memancarkan metrik. `OnFailureDestinationDeliveredEventCount`
  - Untuk `DestinationDeliveryFailures` metrik, Lambda memancarkan 1.
  - Untuk `DroppedEventCount` metrik, Lambda memancarkan angka yang sama dengan jumlah peristiwa yang gagal pengiriman.

- `DeletedEventCount`— Jumlah peristiwa yang berhasil dihapus Lambda setelah diproses. Jika Lambda mencoba menghapus peristiwa tetapi gagal, Lambda memancarkan metrik 0. Gunakan metrik ini untuk memastikan bahwa peristiwa yang berhasil diproses dihapus dari sumber acara Anda.
- `CommittedEventCount`— Jumlah acara yang berhasil dilakukan Lambda setelah diproses. Ini adalah jumlah delta offset komit terakhir dan saat ini dari setiap partisi dalam pemetaan sumber acara Kafka.
- `PollingErrorCount`— Jumlah kesalahan yang Lambda gagal melakukan polling permintaan dari sumber acara. Lambda hanya memancarkan data metrik ini ketika kesalahan terjadi.
- `InvokeErrorCount`— Jumlah kesalahan yang Lambda gagal untuk memanggil Fungsi Acara Anda. Perhatikan pemanggilan adalah catatan dalam batch. Jumlahnya ada di tingkat batch, bukan pada tingkat hitungan rekor. Lambda hanya memancarkan data metrik ini ketika kesalahan terjadi.
- `SchemaRegistryErrorCount`— Jumlah kesalahan yang Lambda gagal mengambil skema atau deserialisasi dengan skema. Lambda hanya memancarkan data metrik ini ketika kesalahan terjadi.
- `CommitErrorCount`— Jumlah kesalahan yang gagal dilakukan Lambda ke cluster Kafka. Lambda hanya memancarkan data metrik ini ketika kesalahan terjadi.
- `MaxOffsetLag`— Maksimum kelambatan offset (perbedaan antara offset terbaru dan yang berkomitmen) di semua partisi dalam pemetaan sumber acara.
- `SumOffsetLag`— Jumlah offset tertinggal di semua partisi dalam pemetaan sumber peristiwa.

Jika pemetaan sumber acara dinonaktifkan, Anda tidak akan menerima metrik pemetaan sumber peristiwa. Anda mungkin juga melihat metrik yang hilang jika CloudWatch atau Lambda mengalami penurunan ketersediaan.

Bekerja dengan log Fungsi Peristiwa Lambda

Untuk membantu Anda memecahkan masalah kegagalan, AWS Lambda secara otomatis memantau Fungsi Acara Lambda atas nama Anda. Anda dapat melihat log untuk Lambda Event Functions menggunakan konsol Lambda, CloudWatch konsol, AWS Command Line Interface (AWS CLI), API. CloudWatch Anda juga dapat mengonfigurasi Lambda untuk mengirim log ke Amazon S3 dan Firehose.

Selama peran eksekusi Fungsi Acara Anda memiliki izin yang diperlukan, Lambda menangkap log untuk semua permintaan yang ditangani oleh Fungsi Peristiwa Anda dan mengirimkannya ke CloudWatch Amazon Logs, yang merupakan tujuan default. Anda juga dapat menggunakan konsol Lambda untuk mengonfigurasi Amazon S3 atau Firehose sebagai tujuan pencatatan.

- CloudWatch Log adalah tujuan logging default untuk Lambda Event Functions. CloudWatch Log menyediakan kemampuan melihat dan menganalisis log secara real-time, dengan dukungan untuk membuat metrik dan alarm berdasarkan data log Anda.
- Amazon S3 ekonomis untuk penyimpanan jangka panjang, dan layanan seperti Athena dapat digunakan untuk menganalisis log. Latensi biasanya lebih tinggi.
- Firehose menawarkan streaming log terkelola ke berbagai tujuan. Jika Anda perlu mengirim log ke AWS layanan lain (misalnya, OpenSearch Service atau Redshift Data API) atau platform pihak ketiga (seperti Datadog, New Relic, atau Splunk), Firehose menyederhanakan proses tersebut dengan menyediakan integrasi pra-bangun. Anda juga dapat melakukan streaming ke titik akhir HTTP khusus tanpa menyiapkan infrastruktur tambahan.

Memilih tujuan layanan untuk mengirim log

Pertimbangkan faktor kunci berikut saat memilih layanan tujuan untuk log Fungsi Peristiwa:

- Manajemen biaya bervariasi menurut layanan. Amazon S3 biasanya menyediakan opsi paling ekonomis untuk penyimpanan jangka panjang, sementara CloudWatch Log menyediakan kemampuan untuk melihat log, memproses log, dan mengatur peringatan secara real time. Biaya Firehose mencakup layanan streaming dan biaya yang terkait dengan apa yang Anda konfigurasikan untuk streaming.
- Kemampuan analisis berbeda di seluruh layanan. CloudWatch Log unggul dalam pemantauan real-time dan terintegrasi secara native dengan CloudWatch fitur lain, seperti Logs Insights dan Live Tail. Amazon S3 bekerja dengan baik dengan alat analisis seperti Athena dan dapat berintegrasi dengan berbagai layanan, meskipun mungkin memerlukan pengaturan tambahan. Firehose

menyederhanakan streaming langsung ke AWS layanan tertentu (seperti Service OpenSearch dan Redshift Data API) dan mendukung platform pihak ketiga (seperti Datadog dan Splunk) dengan menyediakan integrasi pra-bangun, berpotensi mengurangi pekerjaan konfigurasi.

- Pengaturan dan kemudahan penggunaan bervariasi menurut layanan. CloudWatch Log adalah tujuan log default - ia bekerja segera tanpa konfigurasi tambahan dan menyediakan tampilan dan analisis log langsung melalui konsol. CloudWatch Jika Anda memerlukan log yang dikirim ke Amazon S3, Anda harus melakukan beberapa pengaturan awal di konsol Lambda dan mengonfigurasi izin bucket. Jika Anda memerlukan log yang dikirim langsung ke layanan seperti OpenSearch Layanan atau platform analitik pihak ketiga, Firehose dapat menyederhanakan proses tersebut.

Mengkonfigurasi tujuan log

AWS Lambda mendukung beberapa tujuan untuk log Fungsi Acara Anda. Panduan ini menjelaskan tujuan pencatatan yang tersedia dan membantu Anda memilih opsi yang tepat untuk kebutuhan Anda. Terlepas dari tujuan yang Anda pilih, Lambda menyediakan opsi untuk mengontrol format log, pemfilteran, dan pengiriman.

Lambda mendukung format JSON dan teks biasa untuk log Fungsi Acara Anda. Log terstruktur JSON memberikan kemampuan pencarian yang ditingkatkan dan memungkinkan analisis otomatis, sementara log teks biasa menawarkan kesederhanaan dan berpotensi mengurangi biaya penyimpanan. Anda dapat mengontrol log mana yang dikirim Lambda ke tujuan yang Anda pilih dengan mengonfigurasi level log untuk log sistem dan aplikasi. Pemfilteran membantu Anda mengelola biaya penyimpanan dan membuatnya lebih mudah untuk menemukan entri log yang relevan selama debugging.

Untuk petunjuk penyiapan terperinci untuk setiap tujuan, lihat bagian berikut:

- Mengirim log Fungsi Peristiwa Lambda ke Log CloudWatch
- Mengirim log Fungsi Peristiwa Lambda ke Firehose
- Mengirim log Fungsi Peristiwa Lambda ke Amazon S3

Mengkonfigurasi kontrol logging lanjutan untuk Lambda Event Functions

Untuk memberi Anda kontrol lebih besar atas bagaimana log Fungsi Peristiwa Anda ditangkap, diproses, dan dikonsumsi, Lambda menawarkan opsi konfigurasi logging berikut:

- Format log - pilih antara teks biasa dan format JSON terstruktur untuk log Fungsi Acara Anda.
- Tingkat log - untuk log terstruktur JSON, pilih tingkat detail log yang dikirim Lambda CloudWatch ke, `FATAL` seperti,,`ERROR`,, `WARN` `INFO`dan. `DEBUG` `TRACE`
- Grup log - pilih grup CloudWatch log yang dikirimkan oleh Fungsi Acara Anda.

Untuk mempelajari lebih lanjut tentang mengonfigurasi kontrol logging lanjutan, lihat bagian berikut:

- Mengkonfigurasi JSON dan format log teks biasa
- Log-level penyaringan
- Mengkonfigurasi grup CloudWatch log

Mengkonfigurasi JSON dan format log teks biasa

Menangkap output log Anda sebagai pasangan nilai kunci JSON membuatnya lebih mudah untuk mencari dan memfilter saat men-debug Fungsi Acara Anda. Dengan log berformat JSON, Anda juga dapat menambahkan tag dan informasi kontekstual ke log Anda. Ini dapat membantu Anda melakukan analisis otomatis volume besar data log. Kecuali alur kerja pengembangan Anda bergantung pada alat yang ada yang menggunakan log Lambda dalam teks biasa, kami sarankan Anda memilih JSON untuk format log Anda.

> **Instans Terkelola Lambda**
>
> Instans Terkelola Lambda hanya mendukung format log JSON. Saat Anda membuat fungsi Instans Terkelola, Lambda secara otomatis mengonfigurasi format log ke JSON dan Anda tidak dapat mengubahnya menjadi teks biasa. Untuk informasi selengkapnya tentang Instans Terkelola, lihatthe section called “Instans Terkelola”.

Untuk semua runtime terkelola Lambda, Anda dapat memilih apakah log sistem Fungsi Acara dikirim ke CloudWatch Log dalam teks biasa yang tidak terstruktur atau format JSON. Log sistem adalah log yang dihasilkan Lambda dan kadang-kadang dikenal sebagai log peristiwa platform.

Untuk runtime yang didukung, saat Anda menggunakan salah satu metode logging bawaan yang didukung, Lambda juga dapat menampilkan log aplikasi Fungsi Acara Anda (log yang dihasilkan kode Fungsi Peristiwa Anda) dalam format JSON terstruktur. Saat Anda mengonfigurasi format log Fungsi Acara untuk runtime ini, konfigurasi yang Anda pilih berlaku untuk log sistem dan aplikasi.

Untuk runtime yang didukung, jika Fungsi Acara menggunakan pustaka atau metode logging yang didukung, Anda tidak perlu membuat perubahan apa pun pada kode yang ada untuk Lambda untuk menangkap log di JSON terstruktur.

> **Note**
>
> Menggunakan pemformatan log JSON menambahkan metadata tambahan dan mengkodekan pesan log sebagai objek JSON yang berisi serangkaian pasangan nilai kunci. Karena itu, ukuran pesan log Fungsi Acara Anda dapat meningkat.

Runtime dan metode logging yang didukung

Lambda saat ini mendukung opsi untuk menampilkan log aplikasi terstruktur JSON untuk runtime berikut.

Bahasa	Versi yang didukung
Java	Semua runtime Java kecuali Java 8 di Amazon Linux 1
.NET	.NET 8 dan yang lebih baru
Node.js	Node.js 16 dan kemudian
Python	Python 3.8 dan yang lebih baru
Ruby	Ruby 4.0 dan yang lebih baru
Karat	n/a

Agar Lambda dapat mengirim log aplikasi Fungsi Acara Anda ke CloudWatch dalam format JSON terstruktur, Fungsi Acara Anda harus menggunakan alat logging bawaan berikut untuk mengeluarkan log:

- Java: `LambdaLogger` Logger atau Log4j2. Untuk informasi selengkapnya, lihat the section called “Pencatatan log”.
- .NET: `ILambdaLogger` Contoh pada objek konteks. Untuk informasi selengkapnya, lihat Log dan pantau Fungsi Acara C# Lambda.

- Node.js- Metode konsol`console.trace`,`console.debug`,`console.log`,`console.info`,`console.error`, dan`console.warn`. Untuk informasi selengkapnya, lihat the section called “Pencatatan log”.
- Python: Pustaka Python standar. `logging` Untuk informasi selengkapnya, lihat the section called “Pencatatan log”.
- Ruby: Perpustakaan Ruby `Logger` standar. Untuk informasi selengkapnya, lihat the section called “Pencatatan log”.
- Karat: `tracing` Peti. Untuk informasi selengkapnya, lihat the section called “Pencatatan log”.

Untuk runtime Lambda terkelola lainnya, Lambda saat ini hanya mendukung pengambilan log sistem dalam format JSON terstruktur. Namun, Anda masih dapat menangkap log aplikasi dalam format JSON terstruktur dalam runtime apa pun dengan menggunakan alat logging seperti Powertools untuk keluaran keluaran log AWS Lambda yang diformat JSON.

Format log default

Saat ini, format log default untuk semua runtime Lambda adalah teks biasa. Untuk Instans Terkelola Lambda, format log selalu JSON dan tidak dapat diubah.

Jika Anda sudah menggunakan pustaka logging seperti Powertools AWS Lambda untuk menghasilkan log Fungsi Acara dalam format terstruktur JSON, Anda tidak perlu mengubah kode jika memilih pemformatan log JSON. Lambda tidak menyandikan ganda log apa pun yang sudah dikodekan JSON, sehingga log aplikasi Fungsi Acara Anda terus ditangkap seperti sebelumnya.

Format JSON untuk log sistem

Saat Anda mengonfigurasi format log Fungsi Acara Anda sebagai JSON, setiap item log sistem (peristiwa platform) ditangkap sebagai objek JSON yang berisi pasangan nilai kunci dengan kunci berikut:

- `"time"`- waktu pesan log dihasilkan
- `"type"`- jenis acara yang dicatat
- `"record"`- isi dari output log

Format `"record"` nilai bervariasi sesuai dengan jenis peristiwa yang dicatat. Untuk mengetahui informasi selengkapnya, lihat the section called “Jenis objek `Acara` API Telemetri”. Untuk informasi

selengkapnya tentang tingkat log yang ditetapkan ke peristiwa log sistem, lihatthe section called "Pemetaan peristiwa tingkat log sistem".

Sebagai perbandingan, dua contoh berikut menunjukkan output log yang sama dalam format teks biasa dan JSON terstruktur. Perhatikan bahwa dalam kebanyakan kasus, peristiwa log sistem berisi lebih banyak informasi saat output dalam format JSON daripada saat output dalam teks biasa.

Example teks biasa:

```
2024-03-13 18:56:24.046000 fbe8c1   INIT_START  Runtime Version:
 python:3.12.v18  Runtime Version ARN: arn:aws:lambda:eu-
west-1::runtime:edb5a058bfa782cb9cedc6d534ac8b8c193bc28e9a9879d9f5ebaaf619cd0fc0
```

Example JSON terstruktur:

```
{
  "time": "2024-03-13T18:56:24.046Z",
  "type": "platform.initStart",
  "record": {
    "initializationType": "on-demand",
    "phase": "init",
    "runtimeVersion": "python:3.12.v18",
    "runtimeVersionArn": "arn:aws:lambda:eu-
west-1::runtime:edb5a058bfa782cb9cedc6d534ac8b8c193bc28e9a9879d9f5ebaaf619cd0fc0"
  }
}
```

> **Note**
>
> the section called "API Telemetri"Selalu memancarkan acara platform seperti `START` dan `REPORT` dalam format JSON. Mengonfigurasi format log sistem yang dikirim Lambda tidak memengaruhi perilaku API CloudWatch Telemetri Lambda.

Format JSON untuk log aplikasi

Saat Anda mengonfigurasi format log Fungsi Acara Anda sebagai JSON, keluaran log aplikasi yang ditulis menggunakan pustaka dan metode logging yang didukung ditangkap sebagai objek JSON yang berisi pasangan nilai kunci dengan kunci berikut.

- `"timestamp"`- waktu pesan log dihasilkan
- `"level"`- tingkat log yang ditetapkan untuk pesan
- `"message"`- isi pesan log
- `"requestId"`(Python, .NET, dan Node.js) atau `"AWSrequestId"` (Java) - ID permintaan unik untuk pemanggilan Fungsi Acara

Bergantung pada runtime dan metode logging yang digunakan fungsi Anda, objek JSON ini mungkin juga berisi pasangan kunci tambahan. Misalnya, dalam Node.js, jika fungsi Anda menggunakan `console` metode untuk mencatat objek kesalahan menggunakan beberapa argumen, objek JSON berisi pasangan nilai kunci tambahan dengan kuncie`rrorMessage`,`errorType`, dan`stackTrace`. Untuk mempelajari lebih lanjut tentang log berformat JSON di runtime Lambda yang berbeda, lihat, dan. the section called “Pencatatan log” the section called “Pencatatan log” the section called “Pencatatan log”

> **Note**
>
> Kunci yang digunakan Lambda untuk nilai stempel waktu berbeda untuk log sistem dan log aplikasi. Untuk log sistem, Lambda menggunakan kunci `"time"` untuk menjaga konsistensi dengan API Telemetri. Untuk log aplikasi, Lambda mengikuti konvensi runtime dan penggunaan yang didukung. `"timestamp"`

Sebagai perbandingan, dua contoh berikut menunjukkan output log yang sama dalam format teks biasa dan JSON terstruktur.

Example teks biasa:

```
2024-10-27T19:17:45.586Z 79b4f56e-95b1-4643-9700-2807f4e68189 INFO some log message
```

Example JSON terstruktur:

```
{
    "timestamp":"2024-10-27T19:17:45.586Z",
    "level":"INFO",
    "message":"some log message",
    "requestId":"79b4f56e-95b1-4643-9700-2807f4e68189"
}
```

Mengatur format log Event Function

Untuk mengonfigurasi format log untuk Fungsi Acara, Anda dapat menggunakan konsol Lambda atau AWS Command Line Interface ()AWS CLI. Anda juga dapat mengonfigurasi format log Fungsi Acara menggunakan perintah API CreateFunctiondan UpdateFunctionConfigurationLambda, sumber daya AWS Serverless Application Model (AWS SAM) AWS::Serverless: :Function, dan sumber daya: :Function. CloudFormation AWS::Lambda

Mengubah format log Fungsi Acara Anda tidak memengaruhi log yang ada yang disimpan di CloudWatch Log. Hanya log baru yang menggunakan format yang diperbarui.

Jika Anda mengubah format log Fungsi Acara menjadi JSON dan tidak menyetel level log, maka Lambda secara otomatis menyetel level log aplikasi dan level log sistem Fungsi Acara Anda ke INFO. Ini berarti bahwa Lambda hanya mengirimkan output log dari INFO tingkat dan lebih rendah ke Log. CloudWatch Untuk mempelajari selengkapnya tentang pemfilteran tingkat log aplikasi dan sistem, lihat the section called “Log-level penyaringan”

> **Note**
>
> Untuk runtime Python, ketika format log Fungsi Acara Anda diatur ke teks biasa, pengaturan tingkat log default adalah WARN. Ini berarti bahwa Lambda hanya mengirimkan output log dari level WARN dan lebih rendah ke Log. CloudWatch Mengubah format log Fungsi Acara Anda ke JSON mengubah perilaku default ini. Untuk mempelajari lebih lanjut tentang login dengan Python, lihat. the section called “Pencatatan log”

Untuk Node.js fungsi yang memancarkan log format metrik tertanam (EMF), mengubah format log Fungsi Acara Anda CloudWatch menjadi JSON dapat mengakibatkan tidak dapat mengenali metrik Anda.

> **Important**
>
> Jika Event Function Anda menggunakan Powertools for AWS Lambda (TypeScript) atau pustaka klien EMF open-source untuk memancarkan log EMF, perbarui pustaka Powertools dan EMF Anda ke versi terbaru untuk memastikannya dapat terus mengurai log Anda dengan benar. CloudWatch Jika Anda beralih ke format log JSON, kami juga menyarankan Anda melakukan pengujian untuk memastikan kompatibilitas dengan metrik tertanam Fungsi Acara Anda. Untuk saran lebih lanjut tentang fungsi node.js yang memancarkan log EMF, lihat. the

section called “Menggunakan pustaka klien format metrik tertanam (EMF) dengan log JSON terstruktur”

Untuk mengonfigurasi format log Fungsi Peristiwa (konsol)

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih Fungsi Acara
3. Pada halaman konfigurasi Fungsi Acara, pilih Alat pemantauan dan operasi.
4. Di panel konfigurasi Logging, pilih Edit.
5. Di bawah Konten log, untuk format Log pilih Teks atau JSON.
6. Pilih Simpan.

Untuk mengubah format log dari fungsi yang ada (AWS CLI)

- Untuk mengubah format log dari fungsi yang ada, gunakan perintah update-function-configuration. Atur `LogFormat` opsi `LoggingConfig` ke salah satu `JSON` atau`Text`.

```
aws lambda update-function-configuration \
  --function-name myFunction \
  --logging-config LogFormat=JSON
```

Untuk mengatur format log saat Anda membuat fungsi (AWS CLI)

- Untuk mengonfigurasi format log saat Anda membuat fungsi baru, gunakan `--logging-config` opsi dalam perintah create-function. Setel `LogFormat` ke salah satu `JSON` atau`Text`. Contoh perintah berikut menciptakan Node.js fungsi yang output log di JSON terstruktur.

  Jika Anda tidak menentukan format log saat membuat fungsi, Lambda menggunakan format log default untuk versi runtime yang Anda pilih. Untuk informasi tentang format logging default, lihatthe section called “Format log default”.

```
aws lambda create-function \
  --function-name myFunction \
  --runtime nodejs24.x \
  --handler index.handler \
  --zip-file fileb://function.zip \
```

```
  --role arn:aws:iam::123456789012:role/LambdaRole \
  --logging-config LogFormat=JSON
```

Log-level penyaringan

Lambda dapat memfilter log Fungsi Peristiwa Anda sehingga hanya log dengan tingkat detail tertentu atau lebih rendah yang dikirim ke CloudWatch Log. Anda dapat mengonfigurasi pemfilteran tingkat log secara terpisah untuk log sistem Fungsi Acara (log yang dihasilkan Lambda) dan log aplikasi (log yang dihasilkan oleh kode Fungsi Acara Anda).

Untukthe section called “Runtime dan metode logging yang didukung”, Anda tidak perlu membuat perubahan apa pun pada kode Fungsi Acara Anda untuk Lambda untuk memfilter log aplikasi Fungsi Acara Anda.

Untuk semua runtime dan metode logging lainnya, kode Fungsi Acara Anda harus menampilkan peristiwa log ke `stdout` atau `stderr` sebagai objek berformat JSON yang berisi pasangan nilai kunci dengan kunci. `"level"` Misalnya, Lambda menafsirkan output berikut `stdout` sebagai log tingkat DEBUG.

```
print('{"level": "debug", "msg": "my debug log", "timestamp":
 "2024-11-02T16:51:31.587199Z"}')
```

Jika bidang `"level"` nilai tidak valid atau hilang, Lambda menetapkan keluaran log INFO tingkat. Agar Lambda dapat menggunakan bidang stempel waktu, Anda harus menentukan waktu dalam format stempel waktu RFC 3339 yang valid. Jika Anda tidak menyediakan stempel waktu yang valid, Lambda menetapkan log INFO level dan menambahkan stempel waktu untuk Anda.

Saat memberi nama kunci stempel waktu, ikuti konvensi runtime yang Anda gunakan. Lambda mendukung konvensi penamaan yang paling umum digunakan oleh runtime terkelola.

> **Note**
>
> Untuk menggunakan penyaringan tingkat log, Fungsi Acara Anda harus dikonfigurasi untuk menggunakan format log JSON. Format log default untuk semua runtime terkelola Lambda saat ini adalah teks biasa. Untuk mempelajari cara mengonfigurasi format log Fungsi Acara Anda ke JSON, lihatthe section called “Mengatur format log Event Function”.

Untuk log aplikasi (log yang dihasilkan oleh kode fungsi Anda), Anda dapat memilih antara tingkat log berikut.

Tingkat log	Penggunaan standar
TRACE (paling detail)	Informasi paling halus yang digunakan untuk melacak jalur eksekusi kode Anda
DEBUG	Informasi terperinci untuk debugging sistem
INFO	Pesan yang merekam operasi normal fungsi Anda
WARN	Pesan tentang potensi kesalahan yang dapat menyebabkan perilaku tak terduga jika tidak ditangani
Kesalahan	Pesan tentang masalah yang mencegah kode berfungsi seperti yang diharapkan
FATAL (paling detail)	Pesan tentang kesalahan serius yang menyebabkan aplikasi berhenti berfungsi

Ketika Anda memilih tingkat log, Lambda mengirimkan log pada tingkat itu dan lebih rendah ke CloudWatch Log. Misalnya, jika Anda menyetel level log aplikasi Fungsi Acara ke WARN, Lambda tidak mengirim output log di tingkat INFO dan DEBUG. Level log aplikasi default untuk penyaringan log adalah INFO.

Saat Lambda memfilter log aplikasi Fungsi Acara Anda, pesan log tanpa level akan ditetapkan INFO tingkat log.

Untuk log sistem (log yang dihasilkan oleh layanan Lambda), Anda dapat memilih antara tingkat log berikut.

Tingkat log	Penggunaan
DEBUG (paling detail)	Informasi terperinci untuk debugging sistem

Tingkat log	Penggunaan
INFO	Pesan yang merekam operasi normal fungsi Anda
PERINGATAN (paling sedikit detail)	Pesan tentang potensi kesalahan yang dapat menyebabkan perilaku tak terduga jika tidak ditangani

Ketika Anda memilih tingkat log, Lambda mengirimkan log pada tingkat itu dan lebih rendah. Misalnya, jika Anda menyetel level log sistem Fungsi Peristiwa ke INFO, Lambda tidak mengirim output log pada tingkat DEBUG.

Secara default, Lambda menetapkan tingkat log sistem ke INFO. Dengan pengaturan ini, Lambda secara otomatis mengirim `"start"` dan `"report"` mencatat pesan ke. CloudWatch Untuk menerima log sistem yang kurang lebih terperinci, ubah level log menjadi DEBUG atau WARN. Untuk melihat daftar level log tempat Lambda memetakan peristiwa log sistem yang berbeda, lihat. the section called “Pemetaan peristiwa tingkat log sistem”

Mengkonfigurasi penyaringan tingkat log

Untuk mengonfigurasi pemfilteran tingkat log aplikasi dan sistem untuk Fungsi Acara, Anda dapat menggunakan konsol Lambda atau (). AWS Command Line Interface AWS CLIAnda juga dapat mengonfigurasi tingkat log Fungsi Peristiwa menggunakan perintah API CreateFunctiondan UpdateFunctionConfigurationLambda, sumber daya AWS Serverless Application Model (AWS SAM) AWS::Serverless: :Function, dan sumber daya: :Function. CloudFormation AWS::Lambda

Perhatikan bahwa jika Anda menetapkan tingkat log Fungsi Acara dalam kode Anda, pengaturan ini lebih diutamakan daripada pengaturan tingkat log lainnya yang Anda konfigurasikan. Misalnya, jika Anda menggunakan `logging` `setLevel()` metode Python untuk mengatur tingkat logging Fungsi Acara Anda ke INFO, pengaturan ini lebih diutamakan daripada pengaturan WARN yang Anda konfigurasikan menggunakan konsol Lambda.

Untuk mengonfigurasi aplikasi atau tingkat log sistem (konsol) Fungsi Acara yang ada

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih Fungsi Acara.
3. Pada halaman konfigurasi Fungsi Acara, pilih Alat pemantauan dan operasi.

4. Di panel konfigurasi Logging, pilih Edit.
5. Di bawah Konten log, untuk format Log pastikan JSON dipilih.
6. Dengan menggunakan tombol radio, pilih level log Aplikasi yang Anda inginkan dan tingkat log Sistem untuk Fungsi Acara Anda.
7. Pilih Simpan.

Untuk mengkonfigurasi aplikasi atau tingkat log sistem fungsi yang ada (AWS CLI)

- Untuk mengubah tingkat log aplikasi atau sistem dari fungsi yang ada, gunakan perintah update-function-configuration. Gunakan `--logging-config` untuk mengatur `SystemLogLevel` ke salah satu`DEBUG`,`INFO`, atau`WARN`. Setel `ApplicationLogLevel` ke salah satu`DEBUG`,`INFO`,`WARN`,`ERROR`, atau`FATAL`.

```
aws lambda update-function-configuration \
  --function-name myFunction \
  --logging-config LogFormat=JSON,ApplicationLogLevel=ERROR,SystemLogLevel=WARN
```

Untuk mengonfigurasi pemfilteran tingkat log saat Anda membuat fungsi

- Untuk mengonfigurasi penyaringan tingkat log saat Anda membuat fungsi baru, gunakan `--logging-config` untuk mengatur `ApplicationLogLevel` tombol `SystemLogLevel` dan dalam perintah create-function. Setel `SystemLogLevel` ke salah satu`DEBUG`,`INFO`, atau`WARN`. Setel `ApplicationLogLevel` ke salah satu`DEBUG`,`INFO`,`WARN`,`ERROR`, atau`FATAL`.

```
aws lambda create-function \
  --function-name myFunction \
  --runtime nodejs24.x \
  --handler index.handler \
  --zip-file fileb://function.zip \
  --role arn:aws:iam::123456789012:role/LambdaRole \
  --logging-config LogFormat=JSON,ApplicationLogLevel=ERROR,SystemLogLevel=WARN
```

Pemetaan peristiwa tingkat log sistem

Untuk peristiwa log tingkat sistem yang dihasilkan oleh Lambda, tabel berikut mendefinisikan tingkat log yang ditetapkan untuk setiap peristiwa. Untuk mempelajari lebih lanjut tentang peristiwa yang tercantum dalam tabel, lihat the section called “`Event`referensi skema”

Nama peristiwa	Kondisi	Tingkat log yang ditetapkan
InitStart	RuntimeVersion diatur	INFO
InitStart	RuntimeVersion tidak disetel	DEBUG
init RuntimeDone	status=sukses	DEBUG
init RuntimeDone	status! = sukses	WARN
InitReport	InitializationType! = sesuai permintaan	INFO
InitReport	initializationType=Sesuai permintaan	DEBUG
InitReport	status! = sukses	WARN
RestoRestoRestart	RuntimeVersion diatur	INFO
RestoRestoRestart	RuntimeVersion tidak disetel	DEBUG
mengembalikan RuntimeDone	status=sukses	DEBUG
mengembalikan RuntimeDone	status! = sukses	WARN
RestoreReport	status=sukses	INFO
RestoreReport	status! = sukses	WARN
start	-	INFO
RuntimeDone	status=sukses	DEBUG
RuntimeDone	status! = sukses	WARN
laporan	status=sukses	INFO
laporan	status! = sukses	WARN
luas	state=sukses	INFO

Nama peristiwa	Kondisi	Tingkat log yang ditetapkan
luas	negara! = sukses	WARN
LogSubscription	-	INFO
TeleMetrySubscription	-	INFO
LogsDropped	-	WARN

> **Note**
>
> the section called “API Telemetri”Selalu memancarkan set lengkap acara platform. Mengonfigurasi level log sistem yang dikirim Lambda tidak memengaruhi perilaku API CloudWatch Telemetri Lambda.

Pemfilteran tingkat log aplikasi dengan runtime khusus

Saat Anda mengonfigurasi pemfilteran tingkat log aplikasi untuk Fungsi Peristiwa Anda, di belakang layar Lambda menetapkan level log aplikasi di runtime menggunakan variabel lingkungan. `AWS_LAMBDA_LOG_LEVEL` Lambda juga menetapkan format log Event Function Anda menggunakan variabel `AWS_LAMBDA_LOG_FORMAT` lingkungan. Anda dapat menggunakan variabel-variabel ini untuk mengintegrasikan kontrol logging lanjutan Lambda ke dalam runtime kustom.

Agar dapat mengonfigurasi pengaturan logging untuk Fungsi Peristiwa menggunakan runtime khusus dengan konsol Lambda AWS CLI, dan API Lambda, konfigurasikan runtime kustom Anda untuk memeriksa nilai variabel lingkungan ini. Anda kemudian dapat mengonfigurasi logger runtime Anda sesuai dengan format log dan level log yang Anda pilih.

Mengirim log Fungsi Peristiwa Lambda ke Log CloudWatch

Secara default, Lambda secara otomatis menangkap log untuk semua pemanggilan Fungsi Acara dan mengirimkannya ke CloudWatch Log, asalkan peran eksekusi Fungsi Acara Anda memiliki izin yang diperlukan. Log ini, secara default, disimpan dalam grup log bernama/aws/lambda/*<function-name>*. Untuk meningkatkan debugging, Anda dapat menyisipkan pernyataan logging kustom ke dalam kode Anda, yang terintegrasi dengan Lambda dengan mulus dengan Log. CloudWatch Jika diperlukan, Anda dapat mengonfigurasi Fungsi Peristiwa untuk mengirim log ke grup lain

menggunakan konsol Lambda, AWS CLI, atau API Lambda. Lihat the section called “Konfigurasikan log CloudWatch fungsi” untuk mempelajari selengkapnya.

Anda dapat melihat log untuk Fungsi Peristiwa Lambda menggunakan konsol Lambda, konsol, AWS Command Line Interface (AWS CLI), atau API. CloudWatch CloudWatch Untuk informasi lebih lanjut, lihat keMelihat CloudWatch log untuk Fungsi Acara Lambda.

> **Note**
>
> Mungkin diperlukan waktu 5 hingga 10 menit agar log muncul setelah pemanggilan Fungsi Acara.

Izin IAM yang diperlukan

Peran eksekusi Anda memerlukan izin berikut untuk mengunggah log ke CloudWatch Log:

- `logs:CreateLogGroup`
- `logs:CreateLogStream`
- `logs:PutLogEvents`

Untuk mempelajari lebih lanjut, lihat Menggunakan kebijakan berbasis identitas (kebijakan IAM) untuk Log CloudWatch di Panduan Pengguna Amazon. CloudWatch

Anda dapat menambahkan izin CloudWatch Log ini menggunakan kebijakan `AWSLambdaBasicExecutionRole` AWS terkelola yang disediakan oleh Lambda. Untuk menambahkan kebijakan ini ke peran Anda, jalankan perintah berikut:

```
aws iam attach-role-policy --role-name your-role --policy-arn arn:aws:iam::aws:policy/
service-role/AWSLambdaBasicExecutionRole
```

Untuk informasi selengkapnya, lihat the section called “AWS kebijakan terkelola”.

Harga

Tidak ada biaya tambahan untuk menggunakan log Lambda; namun, biaya CloudWatch Log standar berlaku. Untuk informasi lebih lanjut, lihat Harga CloudWatch .

Mengkonfigurasi grup CloudWatch log

Secara default, CloudWatch secara otomatis membuat grup log bernama `/aws/lambda/<function name>` untuk Fungsi Peristiwa Lambda Anda saat pertama kali dipanggil. Untuk mengonfigurasi Fungsi Peristiwa untuk mengirim log ke grup log yang ada, atau untuk membuat grup log baru untuk Fungsi Acara, Anda dapat menggunakan konsol Lambda atau. AWS CLI Anda juga dapat mengonfigurasi grup log kustom menggunakan perintah API CreateFunctiondan UpdateFunctionConfigurationLambda dan sumber daya AWS Serverless Application Model (AWS SAM) AWS: :Serverless: :Function.

Anda dapat mengonfigurasi beberapa Fungsi Peristiwa Lambda untuk mengirim log ke grup log yang sama CloudWatch . Misalnya, Anda dapat menggunakan grup log tunggal untuk menyimpan log untuk semua Fungsi Peristiwa Lambda yang membentuk aplikasi tertentu. Saat Anda menggunakan grup log kustom untuk Fungsi Peristiwa Lambda, aliran log yang dibuat Lambda menyertakan nama Fungsi Acara dan versi Fungsi Peristiwa. Ini memastikan bahwa pemetaan antara pesan log dan Fungsi Peristiwa dipertahankan, bahkan jika Anda menggunakan grup log yang sama untuk beberapa Fungsi Peristiwa.

Format penamaan aliran log untuk grup log kustom mengikuti konvensi ini:

```
YYYY/MM/DD/<function_name>[<function_version>][<execution_environment_GUID>]
```

Perhatikan bahwa saat mengonfigurasi grup log kustom, nama yang Anda pilih untuk grup log harus mengikuti aturan penamaan CloudWatch Log. Selain itu, nama grup log kustom tidak boleh dimulai dengan string`aws/`. Jika Anda membuat grup log kustom dimulai dengan`aws/`, Lambda tidak akan dapat membuat grup log. Sebagai akibatnya, log Event Function Anda tidak akan dikirim ke CloudWatch.

Untuk mengubah grup log fungsi (konsol)

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih Fungsi Acara.
3. Pada halaman konfigurasi Fungsi Acara, pilih Alat pemantauan dan operasi.
4. Di panel konfigurasi Logging, pilih Edit.
5. Di panel grup Logging, untuk grup CloudWatch log, pilih Kustom.
6. Di bawah Grup log kustom, masukkan nama grup CloudWatch log yang Anda inginkan untuk mengirim log ke fungsi Anda. Jika Anda memasukkan nama grup log yang ada, maka fungsi

Anda akan menggunakan grup itu. Jika tidak ada grup log dengan nama yang Anda masukkan, maka Lambda akan membuat grup log baru untuk fungsi Anda dengan nama itu.

Untuk mengubah grup log Fungsi Acara (AWS CLI)

- Untuk mengubah grup log dari Fungsi Peristiwa yang ada, gunakan perintah update-function-configuration.

```
aws lambda update-function-configuration \
  --function-name myFunction \
  --logging-config LogGroup=myLogGroup
```

Untuk menentukan grup log kustom saat Anda membuat Fungsi Acara (AWS CLI)

- Untuk menentukan grup log kustom saat Anda membuat Fungsi Peristiwa Lambda baru menggunakan AWS CLI, gunakan opsi. `--logging-config` Contoh perintah berikut membuat Node.js Lambda Event Function yang mengirimkan log ke grup log bernama. `myLogGroup`

```
aws lambda create-function \
  --function-name myFunction \
  --runtime nodejs24.x \
  --handler index.handler \
  --zip-file fileb://function.zip \
  --role arn:aws:iam::123456789012:role/LambdaRole \
  --logging-config LogGroup=myLogGroup
```

Izin peran eksekusi

Agar Fungsi Acara Anda mengirim CloudWatch log ke Log, ia harus memiliki PutLogEvents izin log:. Saat Anda mengonfigurasi grup log Fungsi Acara menggunakan konsol Lambda, Lambda menambahkan izin ini ke peran dalam kondisi berikut:

- Tujuan layanan diatur ke CloudWatch Log
- Peran eksekusi Event Function Anda tidak memiliki izin untuk mengunggah CloudWatch log ke Log (tujuan default)

> **Note**
>
> Lambda tidak menambahkan izin Put untuk tujuan log Amazon S3 atau Firehose.

Ketika Lambda menambahkan izin ini, ia memberikan izin Fungsi Acara untuk mengirim log ke grup log Log mana pun CloudWatch .

Untuk mencegah Lambda memperbarui peran eksekusi Fungsi Acara secara otomatis dan mengeditnya secara manual, perluas Izin dan hapus centang Tambahkan izin yang diperlukan.

Saat Anda mengonfigurasi grup log Fungsi Acara menggunakan AWS CLI, Lambda tidak akan menambahkan izin secara otomatis. `logs:PutLogEvents` Tambahkan izin ke peran eksekusi Fungsi Acara Anda jika belum memilikinya. Izin ini termasuk dalam kebijakan yang AWSLambdaBasicExecutionRoledikelola.

CloudWatch logging untuk Instans Terkelola Lambda

Saat menggunakan Instans Terkelola Lambda, ada pertimbangan tambahan untuk mengirim log ke Log: CloudWatch

Persyaratan jaringan VPC

Instans Terkelola Lambda berjalan pada instans EC2 milik pelanggan dalam VPC Anda. Untuk mengirim CloudWatch log ke Log dan trace ke X-Ray, Anda harus memastikan bahwa AWS API ini dapat dirutekan dari VPC Anda. Anda memiliki beberapa pilihan:

- AWS PrivateLink (disarankan): Gunakan AWS PrivateLinkuntuk membuat titik akhir VPC untuk CloudWatch Log dan layanan. X-Ray Ini memungkinkan instans Anda untuk mengakses layanan ini secara pribadi tanpa memerlukan gateway internet atau gateway NAT. Untuk informasi selengkapnya, lihat Menggunakan CloudWatch Log dengan titik akhir VPC antarmuka.
- NAT Gateway: Konfigurasikan gateway NAT untuk memungkinkan akses internet keluar dari subnet pribadi Anda.
- Internet Gateway: Untuk subnet publik, pastikan VPC Anda memiliki gateway internet yang dikonfigurasi.

Jika CloudWatch Log atau X-Ray API tidak dapat dirutekan dari VPC Anda, log dan jejak Fungsi Acara Anda tidak dikirimkan.

Pemanggilan bersamaan dan atribusi log

Lingkungan eksekusi Instans Terkelola Lambda dapat memproses beberapa pemanggilan secara bersamaan. Ketika beberapa pemanggilan berjalan secara bersamaan, entri lognya disisipkan dalam aliran log yang sama. Untuk memfilter dan menganalisis log secara efektif dari pemanggilan bersamaan, Anda harus memastikan setiap entri log menyertakan ID permintaan. AWS

Kami merekomendasikan salah satu pendekatan berikut:

- Gunakan pencatat runtime Lambda default (disarankan): Pustaka logging default yang disediakan oleh runtime terkelola Lambda secara otomatis menyertakan ID permintaan di setiap entri log.
- Menerapkan logging JSON terstruktur: Jika Anda membuat runtime khusus atau memerlukan pencatatan khusus, terapkan JSON-formatted log yang menyertakan ID permintaan di setiap entri. Instans Terkelola Lambda hanya mendukung format log JSON. Sertakan `requestId` bidang di log JSON Anda untuk mengaktifkan pemfilteran berdasarkan pemanggilan:

```
{
  "timestamp": "2025-01-15T10:30:00.000Z",
  "level": "INFO",
  "requestId": "a1b2c3d4-e5f6-7890-abcd-ef1234567890",
  "message": "Processing request"
}
```

Dengan atribusi ID permintaan, Anda dapat memfilter entri CloudWatch log Log untuk pemanggilan tertentu menggunakan CloudWatch kueri Wawasan Log. Contoh:

```
fields @timestamp, @message
| filter requestId = "a1b2c3d4-e5f6-7890-abcd-ef1234567890"
| sort @timestamp asc
```

Untuk informasi selengkapnya tentang persyaratan pencatatan Instans Terkelola Lambda, lihat. Memahami lingkungan eksekusi Instans Terkelola Lambda

Melihat CloudWatch log untuk Fungsi Acara Lambda

Anda dapat melihat CloudWatch log Amazon untuk Fungsi Peristiwa Lambda menggunakan konsol Lambda, CloudWatch konsol, atau (). AWS Command Line Interface AWS CLI Ikuti petunjuk di bagian berikut untuk mengakses log Fungsi Acara Anda.

Streaming log Fungsi Acara dengan CloudWatch Log Live Tail

Amazon CloudWatch Logs Live Tail membantu Anda memecahkan masalah Fungsi Acara dengan cepat dengan menampilkan daftar streaming peristiwa log baru secara langsung di konsol Lambda. Anda dapat melihat dan memfilter log yang dicerna dari Fungsi Peristiwa Lambda Anda secara real time, membantu Anda mendeteksi dan menyelesaikan masalah dengan cepat.

> **Note**
>
> Sesi Live Tail dikenakan biaya berdasarkan waktu penggunaan sesi, per menit. Untuk informasi selengkapnya tentang harga, lihat CloudWatch Harga Amazon.

Membandingkan Live Tail dan --log-type Tail

Ada beberapa perbedaan antara CloudWatch Logs Live Tail dan opsi LogType: Tail di Lambda API (`--log-type Tail`di AWS CLI):

- `--log-type Tail`mengembalikan hanya 4 KB pertama dari log pemanggilan. Live Tail tidak membagikan batas ini, dan dapat menerima hingga 500 peristiwa log per detik.
- `--log-type Tail`menangkap dan mengirim log dengan respons, yang dapat memengaruhi latensi respons Fungsi Peristiwa. Live Tail tidak memengaruhi latensi respons Fungsi Peristiwa.
- `--log-type Tail`hanya mendukung pemanggilan sinkron. Live Tail berfungsi untuk pemanggilan sinkron dan asinkron.

> **Note**
>
> Instans Terkelola Lambda tidak mendukung opsi. `--log-type Tail` Gunakan CloudWatch Log Live Tail atau kueri CloudWatch Log secara langsung untuk melihat log untuk fungsi Instans Terkelola.

Izin

Izin berikut diperlukan untuk memulai dan menghentikan sesi CloudWatch Log Live Tail:

- `logs:DescribeLogGroups`
- `logs:StartLiveTail`

- `logs:StopLiveTail`

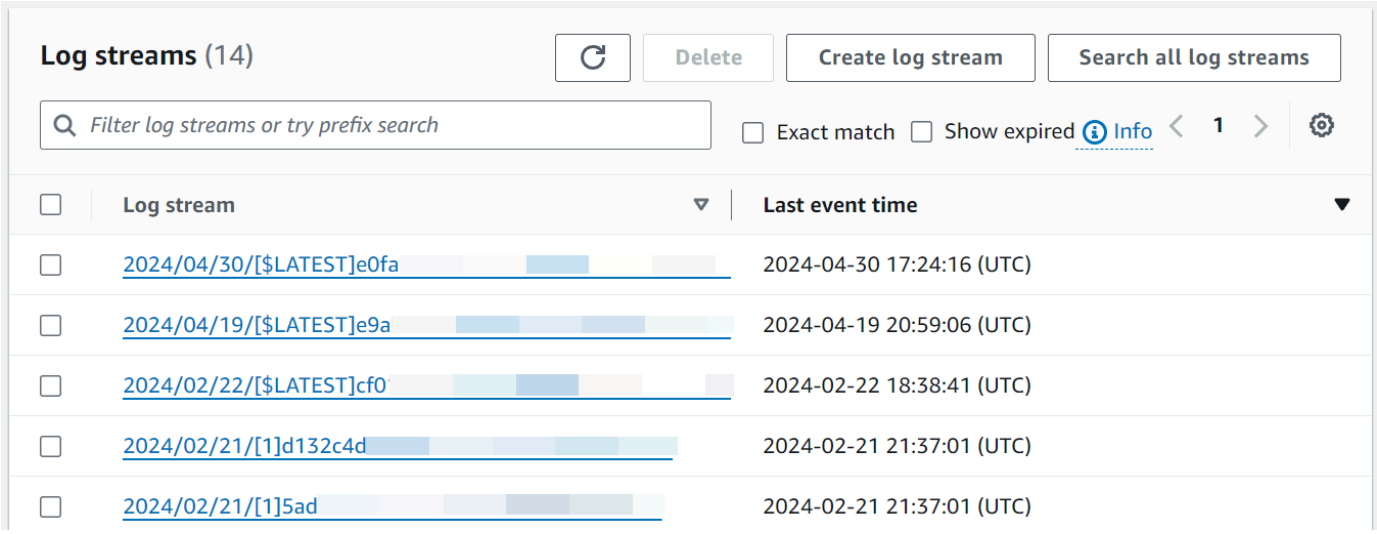
Mulai sesi Live Tail di konsol Lambda

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih nama Fungsi Acara.
3. Pilih tab Uji.
4. Di panel acara Uji, pilih CloudWatch Log Live Tail.
5. Untuk Pilih grup log, grup log Fungsi Acara dipilih secara default. Anda dapat memilih hingga lima grup log sekaligus.
6. (Opsional) Untuk menampilkan hanya peristiwa log yang berisi kata-kata tertentu atau string lainnya, masukkan kata atau string di Tambahkan pola filter kotak. Bidang filter peka huruf besar/kecil. Anda dapat menyertakan beberapa istilah dan operator pola di bidang ini, termasuk ekspresi reguler (regex). Untuk informasi selengkapnya tentang sintaks pola, lihat Filter sintaks pola. di Panduan Pengguna Amazon CloudWatch Logs.
7. Pilih Mulai. Peristiwa log yang cocok mulai muncul di jendela.
8. Untuk menghentikan sesi Live Tail, pilih Stop.

> **Note**
>
> Sesi Live Tail otomatis berhenti setelah 15 menit tidak aktif atau saat sesi konsol Lambda habis waktu.

Akses log Fungsi Peristiwa menggunakan konsol

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih Fungsi Acara.
3. Pilih tab Pantau.
4. Pilih Lihat CloudWatch log untuk membuka CloudWatch konsol.
5. Gulir ke bawah dan pilih aliran Log untuk pemanggilan fungsi yang ingin Anda lihat.



Setiap instance dari Lambda Event Function memiliki aliran log khusus. Jika Fungsi Peristiwa meningkat, setiap instance bersamaan memiliki aliran lognya sendiri. Setiap kali lingkungan eksekusi baru dibuat sebagai respons terhadap pemanggilan, ini menghasilkan aliran log baru. Konvensi penamaan untuk aliran log adalah:

```
YYYY/MM/DD[Function version][Execution environment GUID]
```

Lingkungan eksekusi tunggal menulis ke aliran log yang sama selama masa pakainya. Aliran log berisi pesan dari lingkungan eksekusi itu dan juga output apa pun dari kode Fungsi Peristiwa Lambda Anda. Setiap pesan diberi cap waktu, termasuk log kustom Anda. Bahkan jika Fungsi Acara Anda tidak mencatat output apa pun dari kode Anda, ada tiga pernyataan log minimal yang dihasilkan per pemanggilan (START, END, dan REPORT):

```
2020-10-08T15:52:11.447-04:00	START RequestId: 345a1711-d325-4af6-b01f-b0648975743f Version: $LATEST
START RequestId: 345a1711-d325-4af6-b01f-b0648975743f Version: $LATEST
Copy

2020-10-08T15:52:12.452-04:00	END RequestId: 345a1711-d325-4af6-b01f-b0648975743f
END RequestId: 345a1711-d325-4af6-b01f-b0648975743f
Copy

2020-10-08T15:52:12.452-04:00	REPORT RequestId: 345a1711-d325-4af6-b01f-b0648975743f Duration: 1004.58 ms Billed Duration: 1100 ms Memory Size: 1...
REPORT RequestId: 345a1711-d325-4af6-b01f-b0648975743f  Duration: 1004.58 ms    Billed Duration: 1100 ms        Memory Size: 128 MB     Max
Memory Used: 64 MB      Init Duration: 295.85 ms
Copy
```

Log ini menunjukkan:

- RequestId— ini adalah ID unik yang dihasilkan per permintaan. Jika Fungsi Peristiwa Lambda mencoba ulang permintaan, ID ini tidak berubah dan muncul di log untuk setiap percobaan ulang berikutnya.

- Start/End— bookmark ini satu doa, jadi setiap baris log di antara ini termasuk dalam doa yang sama.
- Durasi — total waktu pemanggilan untuk fungsi handler, tidak termasuk kode. `INIT`
- Durasi Tagihan - menerapkan logika pembulatan untuk tujuan penagihan.
- Ukuran Memori — jumlah memori yang dialokasikan ke Fungsi Acara.
- Max Memory Used — jumlah maksimum memori yang digunakan selama pemanggilan.
- Init Duration — waktu yang dibutuhkan untuk menjalankan `INIT` bagian kode, di luar handler utama.

Akses log dengan AWS CLI

AWS CLI Ini adalah alat sumber terbuka yang dapat Anda gunakan untuk berinteraksi dengan AWS layanan menggunakan perintah di shell baris perintah Anda. Untuk menyelesaikan langkah-langkah di bagian ini, Anda harus memiliki AWS CLI versi 2.

Anda dapat menggunakan AWS CLI untuk mengambil log untuk invokasi menggunakan opsi perintah `--log-type`. Respons berisi bidang `LogResult` yang memuat hingga 4 KB log berkode base64 dari invokasi.

Example mengambil ID log

Contoh berikut menunjukkan cara mengambil ID log dari `LogResult` untuk fungsi bernama `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

Anda akan melihat output berikut:

```
{
    "StatusCode": 200,
    "LogResult":
 "U1RBUlQgUmVxdWVzdElkOiA4N2QwNDRiOC1mMTU0LTExZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "ExecutedVersion": "$LATEST"
}
```

Example mendekode log

Pada prompt perintah yang sama, gunakan utilitas `base64` untuk mendekodekan log. Contoh berikut menunjukkan cara mengambil log berkode base64 untuk `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail \
--query 'LogResult' --output text --cli-binary-format raw-in-base64-out | base64 --
decode
```

cli-binary-formatOpsi ini diperlukan jika Anda menggunakan AWS CLI versi 2. Untuk menjadikan ini pengaturan default, jalankan`aws configure set cli-binary-format raw-in-base64-out`. Untuk informasi selengkapnya, lihat opsi baris perintah global yang AWS CLI didukung di Panduan AWS Command Line Interface Pengguna untuk Versi 2.

Anda akan melihat output berikut:

```
START RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8 Version: $LATEST
"AWS_SESSION_TOKEN": "AgoJb3JpZ2luX2VjELj...", "_X_AMZN_TRACE_ID": "Root=1-5d02e5ca-
f5792818b6fe8368e5b51d50;Parent=191db58857df8395;Sampled=0"",ask/lib:/opt/lib",
END RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8
REPORT RequestId: 57f231fb-1730-4395-85cb-4f71bd2b87b8  Duration: 79.67 ms      Billed
 Duration: 80 ms         Memory Size: 128 MB     Max Memory Used: 73 MB
```

Utilitas `base64` tersedia di Linux, macOS, dan Ubuntu pada Windows. Pengguna macOS mungkin harus menggunakan `base64 -D`.

Example Skrip get-logs.sh

Pada prompt perintah yang sama, gunakan script berikut untuk mengunduh lima peristiwa log terakhir. Skrip menggunakan `sed` untuk menghapus kutipan dari file output, dan akan tidur selama 15 detik untuk memberikan waktu agar log tersedia. Output mencakup respons dari Lambda dan output dari perintah `get-log-events`.

Salin konten dari contoh kode berikut dan simpan dalam direktori proyek Lambda Anda sebagai `get-logs.sh`.

cli-binary-formatOpsi ini diperlukan jika Anda menggunakan AWS CLI versi 2. Untuk menjadikan ini pengaturan default, jalankan`aws configure set cli-binary-format raw-in-base64-out`. Untuk informasi selengkapnya, lihat opsi baris perintah global yang AWS CLI didukung di Panduan AWS Command Line Interface Pengguna untuk Versi 2.

```
#!/bin/bash
aws lambda invoke --function-name my-function --cli-binary-format raw-in-base64-out --
payload '{"key": "value"}' out
```

```
sed -i'' -e 's/"//g' out
sleep 15
aws logs get-log-events --log-group-name /aws/lambda/my-function --log-stream-
name stream1 --limit 5
```

Example macOS dan Linux (khusus)

Dalam prompt perintah yang sama, pengguna macOS dan Linux mungkin perlu menjalankan perintah berikut untuk memastikan skrip dapat dieksekusi.

```
chmod -R 755 get-logs.sh
```

Example mengambil lima log acara terakhir

Pada prompt perintah yang sama, gunakan skrip berikut untuk mendapatkan lima log acara terakhir.

```
./get-logs.sh
```

Anda akan melihat output berikut:

```
{
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}
{
    "events": [
        {
            "timestamp": 1559763003171,
            "message": "START RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf Version:
 $LATEST\n",
            "ingestionTime": 1559763003309
        },
        {
            "timestamp": 1559763003173,
            "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tENVIRONMENT VARIABLES\r{\r  \"AWS_LAMBDA_FUNCTION_VERSION\": \"$LATEST\",
\r ...",
            "ingestionTime": 1559763018353
        },
        {
            "timestamp": 1559763003173,
```

```
            "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tINFO\tEVENT\r{\r  \"key\": \"value\"\r}\n",
            "ingestionTime": 1559763018353
        },
        {
            "timestamp": 1559763003218,
            "message": "END RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf\n",
            "ingestionTime": 1559763018353
        },
        {
            "timestamp": 1559763003218,
            "message": "REPORT RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tDuration: 26.73 ms\tBilled Duration: 27 ms \tMemory Size: 128 MB\tMax Memory Used: 75
 MB\t\n",
            "ingestionTime": 1559763018353
        }
    ],
    "nextForwardToken": "f/34783877304859518393868359594929986069206639495374241795",
    "nextBackwardToken": "b/34783877303811383369537420289090800615709599058929582080"
}
```

Mengurai log dan logging terstruktur

Dengan Wawasan CloudWatch Log, Anda dapat mencari dan menganalisis data log menggunakan sintaks kueri khusus. Ini melakukan kueri pada beberapa grup log dan menyediakan pemfilteran yang kuat menggunakan pencocokan pola ekspresi glob dan reguler.

Anda dapat memanfaatkan kemampuan ini dengan menerapkan logging terstruktur di Fungsi Peristiwa Lambda Anda. Pencatatan terstruktur mengatur log Anda ke dalam format yang telah ditentukan sebelumnya, sehingga lebih mudah untuk melakukan kueri. Menggunakan level log merupakan langkah pertama yang penting dalam menghasilkan log ramah filter yang memisahkan pesan informasi dari peringatan atau kesalahan. Misalnya, perhatikan Node.js kode berikut:

```
exports.handler = async (event) => {
    console.log("console.log - Application is fine")
    console.info("console.info - This is the same as console.log")
    console.warn("console.warn - Application provides a warning")
    console.error("console.error - An error occurred")
}
```

File CloudWatch log yang dihasilkan berisi bidang terpisah yang menentukan tingkat log:

```
START RequestId: 99d91f9b-2dff-40ad-b9c8-664020094109 Version: $LATEST
2020-10-22T12:21:28.268Z        99d91f9b-2dff-40ad-b9c8-664020094109    INFO    console.log - Application is fine
2020-10-22T12:21:28.268Z        99d91f9b-2dff-40ad-b9c8-664020094109    INFO    console.info - This is the same as console.log
2020-10-22T12:21:28.268Z        99d91f9b-2dff-40ad-b9c8-664020094109    WARN    console.warn - Application provides a warning
2020-10-22T12:21:28.268Z        99d91f9b-2dff-40ad-b9c8-664020094109    ERROR   console.error - An error occurred
END RequestId: 99d91f9b-2dff-40ad-b9c8-664020094109
REPORT RequestId: 99d91f9b-2dff-40ad-b9c8-664020094109  Duration: 3.13 ms       Billed Duration: 100 ms Memory Size: 128 MB
Max Memory Used: 64 MB  Init Duration: 142.18 ms
```

Kueri Wawasan CloudWatch Log kemudian dapat memfilter pada tingkat log. Misalnya, untuk menanyakan kesalahan saja, Anda dapat menggunakan kueri berikut:

```
fields @timestamp, @message
| filter @message like /ERROR/
| sort @timestamp desc
```

Pencatatan terstruktur JSON

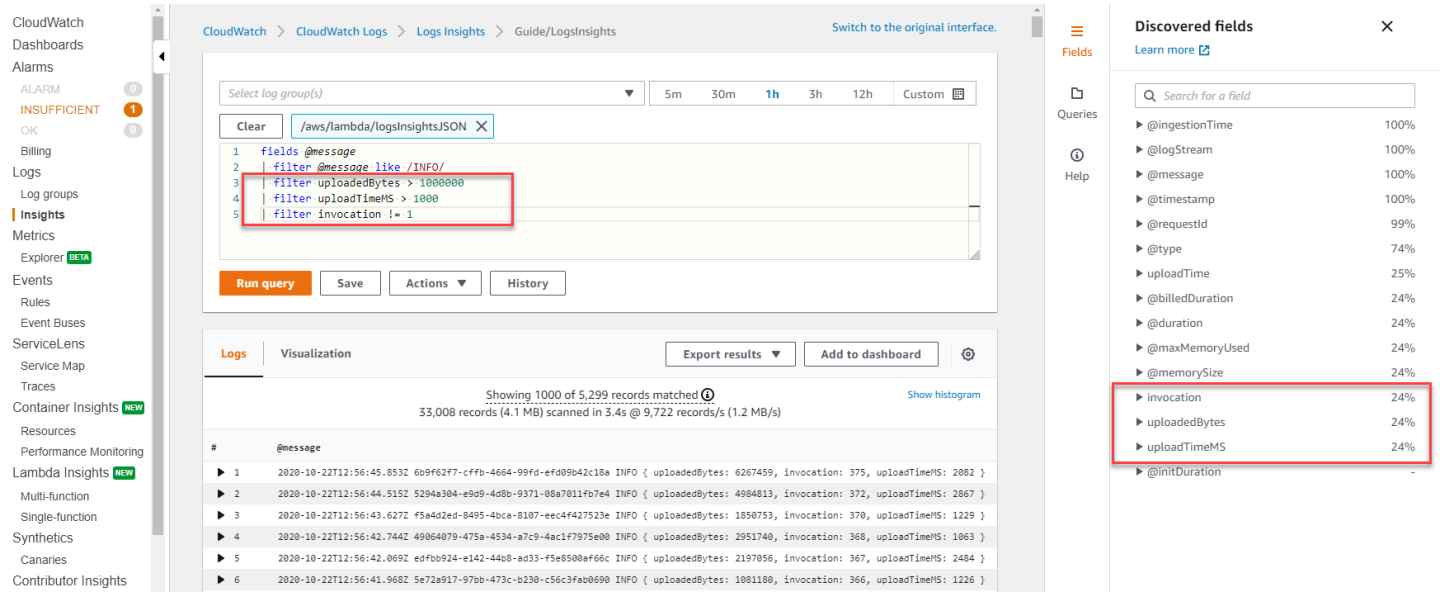
JSON umumnya digunakan untuk menyediakan struktur untuk log aplikasi. Dalam contoh berikut, log telah dikonversi ke JSON untuk menghasilkan tiga nilai yang berbeda:

```
START RequestId: 22fda338-7b46-4c49-9531-efd6e8568480 Version: $LATEST
2020-10-22T11:35:59.216Z        22fda338-7b46-4c49-9531-efd6e8568480    INFO    { uploadedBytes: 5453396,
invocation: 5, uploadTimeMS: 4519 }
END RequestId: 22fda338-7b46-4c49-9531-efd6e8568480
REPORT RequestId: 22fda338-7b46-4c49-9531-efd6e8568480  Duration: 1.25 ms       Billed Duration: 100 ms Memory
Size: 128 MB    Max Memory Used: 65 MB
```

Fitur Wawasan CloudWatch Log secara otomatis menemukan nilai dalam keluaran JSON dan mem-parsing pesan sebagai bidang, tanpa perlu glob kustom atau ekspresi reguler. Dengan menggunakan JSON-structured log, kueri berikut menemukan pemanggilan di mana file yang diunggah lebih besar dari 1 MB, waktu unggah lebih dari 1 detik, dan pemanggilan bukanlah awal yang dingin:

```
fields @message
| filter @message like /INFO/
| filter uploadedBytes > 1000000
| filter uploadTimeMS > 1000
| filter invocation != 1
```

Kueri ini mungkin menghasilkan hasil sebagai berikut:



Bidang yang ditemukan di JSON secara otomatis diisi di menu bidang Ditemukan di sisi kanan. Bidang standar yang dipancarkan oleh layanan Lambda diawali dengan '@', dan Anda dapat melakukan kueri pada bidang ini dengan cara yang sama. Log Lambda selalu menyertakan bidang @timestamp, @logStream, @message, @requestId, @duration, @billedDuration, @type, @maxMemoryUsed,. @memorySize Jika X-Ray diaktifkan untuk Fungsi Peristiwa, log juga menyertakan @xray TraceId dan @xraySegmentId.

Saat sumber AWS peristiwa seperti Amazon S3, Amazon SQS, atau EventBridge Amazon memanggil Fungsi Acara Anda, seluruh acara disediakan ke Fungsi Peristiwa sebagai input objek JSON. Dengan mencatat peristiwa ini di baris pertama Fungsi Peristiwa, Anda kemudian dapat melakukan kueri pada salah satu bidang bersarang menggunakan Wawasan CloudWatch Log.

Pertanyaan Wawasan Berguna

Tabel berikut menunjukkan contoh kueri Wawasan yang dapat berguna untuk memantau Fungsi Peristiwa Lambda.

Deskripsi	Contoh sintaks kueri

100 kesalahan terakhir	`fields Timestamp, LogLevel, Message`
	`| filter LogLevel == "ERR"`
	`| sort @timestamp desc`
	`| limit 100`

100 pemanggilan dengan tagihan tertinggi	`filter @type = "REPORT"`

Deskripsi	Contoh sintaks kueri

	`| fields @requestId, @billedDuration`
	`| sort by @billedDuration desc`
	`| limit 100`

Persentase dingin dimulai dalam total pemanggilan	`filter @type = "REPORT"`
	`| stats sum(strcontains(@message,`
	` "Init Duration"))/count(*) * 100 as`
	`  coldStartPct, avg(@duration)`
	`  by bin(5m)`

Laporan persentil durasi Lambda	`filter @type = "REPORT"`
	`| stats`
	`  avg(@billedDuration) as Average,`
	`  percentile(@billedDuration, 99)`
	` as NinetyNinth,`
	`  percentile(@billedDuration, 95)`
	` as NinetyFifth,`
	`  percentile(@billedDuration, 90)`
	` as Ninetieth`
	`  by bin(30m)`

Laporan persentil penggunaan memori Lambda	`filter @type="REPORT"`
	`| stats avg(@maxMemoryUsed/1024/102`
	`4) as mean_MemoryUsed,`
	`  min(@maxMemoryUsed/1024/1024) as`
	` min_MemoryUsed,`
	`  max(@maxMemoryUsed/1024/1024) as`
	` max_MemoryUsed,`
	`  percentile(@maxMemoryUsed/1`
	`024/1024, 95) as Percentile95`

Pemanggilan menggunakan 100% memori yang ditetapkan	`filter @type = "REPORT" and @maxMemor`
	`yUsed=@memorySize`
	`| stats`
	`  count_distinct(@requestId)`
	`  by bin(30m)`

Deskripsi	Contoh sintaks kueri

Memori rata-rata yang digunakan di seluruh pemanggilan	`avgMemoryUsedPERC,`
	`    avg(@billedDuration) as avgDurati`
	`onMS`
	`    by bin(5m)`

Visualisasi statistik memori	`filter @type = "REPORT"`
	`| stats`
	`    max(@maxMemoryUsed / 1024 / 1024)`
	` as maxMemMB,`
	`    avg(@maxMemoryUsed / 1024 / 1024)`
	` as avgMemMB,`
	`    min(@maxMemoryUsed / 1024 / 1024)`
	` as minMemMB,`
	`    (avg(@maxMemoryUsed / 1024 /`
	` 1024) / max(@memorySize / 1024 /`
	` 1024)) * 100 as avgMemUsedPct,`
	`    avg(@billedDuration) as avgDurati`
	`onMS`
	`    by bin(30m)`

Doa tempat Lambda keluar	`filter @message like /Process exited/` `| stats count() by bin(30m)`
Doa yang habis waktunya	`filter @message like /Task timed out/` `| stats count() by bin(30m)`

Laporan latensi	`filter @type = "REPORT"`
	`| stats avg(@duration), max(@dura`
	`tion), min(@duration)`
	`  by bin(5m)`

Deskripsi	Contoh sintaks kueri
Over-provisioned memori	`filter @type = "REPORT" | stats max(@memorySize / 1024 / 1024) as provisonedMemMB, min(@maxMemoryUsed / 1024 / 1024) as smallestMemReqMB, avg(@maxMemoryUsed / 1024 / 1024) as avgMemUsedMB, max(@maxMemoryUsed / 1024 / 1024) as maxMemUsedMB, provisonedMemMB - maxMemUsedMB as overProvisionedMB`

Visualisasi log dan dasbor

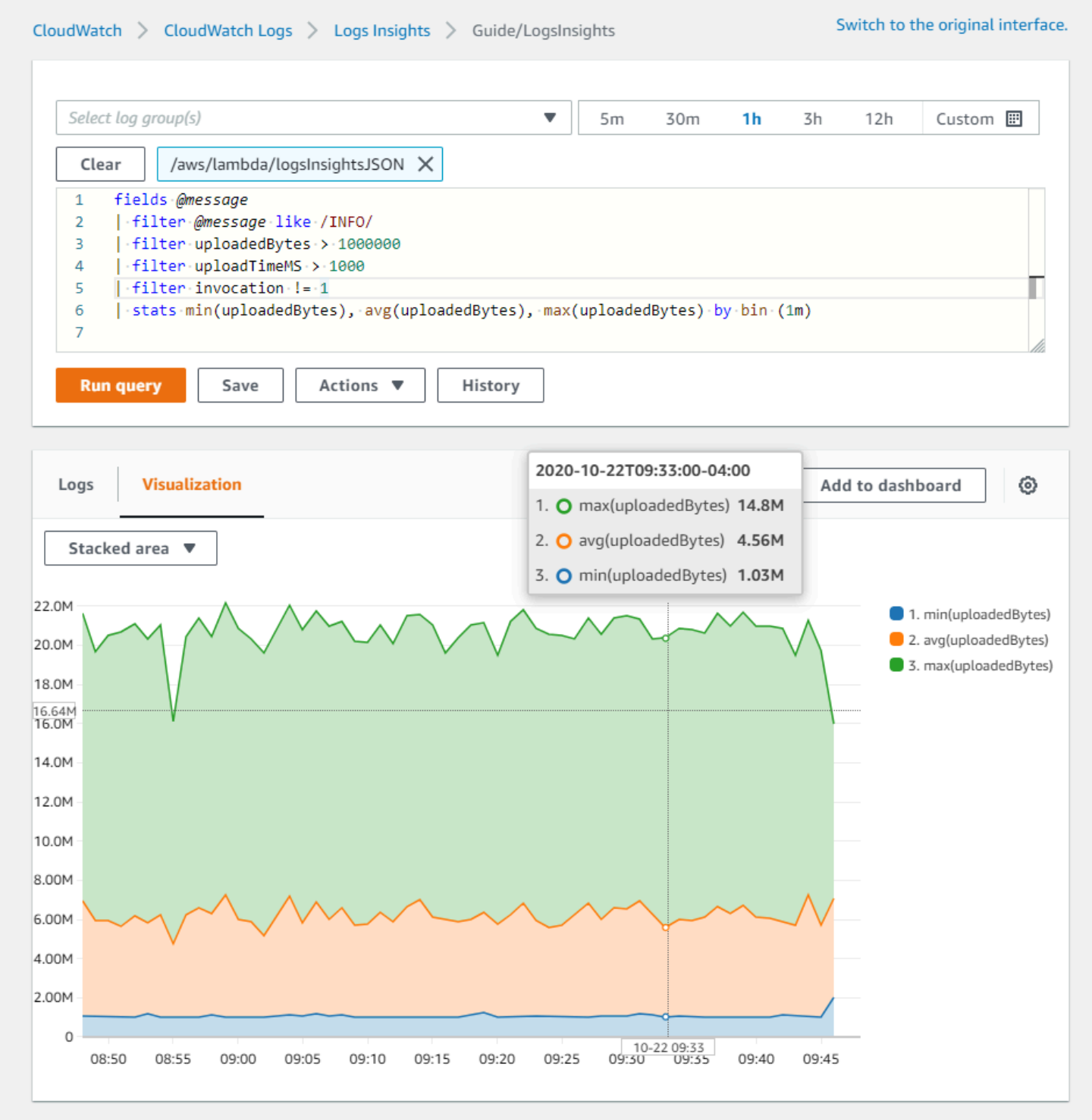
Untuk kueri Wawasan CloudWatch Log apa pun, Anda dapat mengekspor hasilnya ke format penurunan harga atau CSV. Dalam beberapa kasus, mungkin lebih berguna untuk membuat visualisasi dari kueri, asalkan setidaknya ada satu fungsi agregasi. Dengan `stats` fungsi ini, Anda dapat menentukan agregasi dan pengelompokan.

Contoh LogInsightsJSON sebelumnya difilter pada ukuran upload dan waktu upload dan mengecualikan pemanggilan pertama. Ini menghasilkan tabel data. Untuk memantau sistem produksi, mungkin lebih berguna untuk memvisualisasikan ukuran file minimum, maksimum, dan rata-rata untuk menemukan outlier. Untuk melakukan ini, terapkan fungsi statistik dengan agregat yang diperlukan, dan kelompokkan pada nilai waktu seperti setiap menit:

Misalnya, pertimbangkan kueri berikut. Ini adalah contoh kueri yang sama dari the section called “Pencatatan terstruktur JSON” bagian, tetapi dengan fungsi agregasi tambahan:

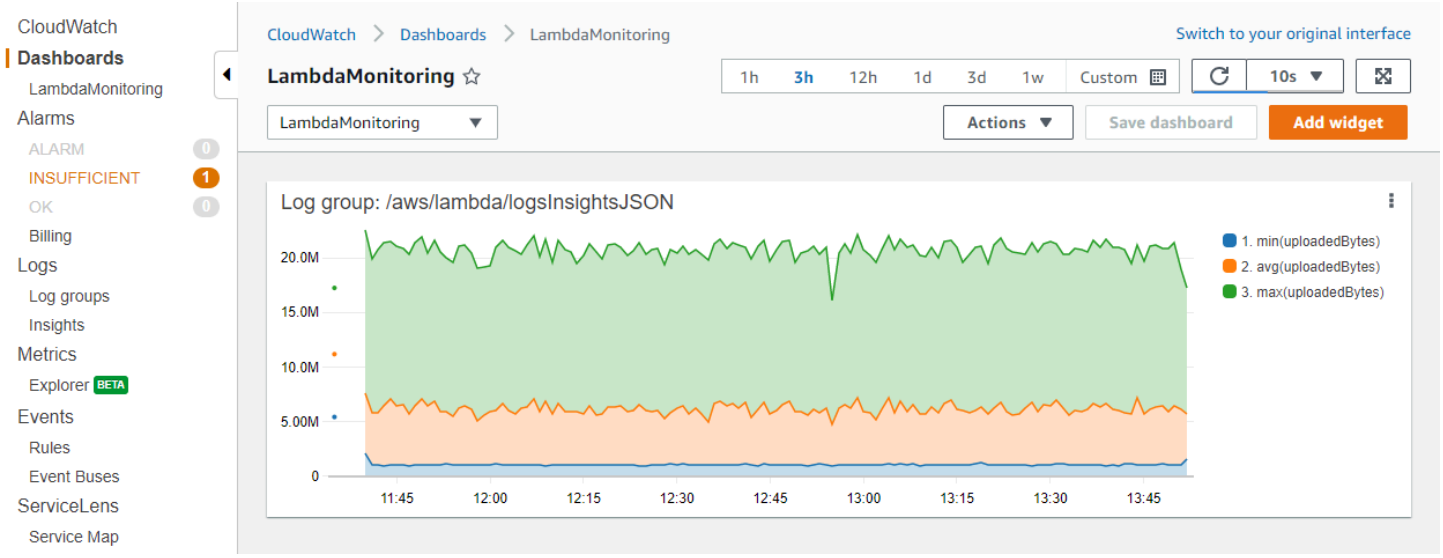
```
fields @message
| filter @message like /INFO/
| filter uploadedBytes > 1000000
| filter uploadTimeMS > 1000
| filter invocation != 1
| stats min(uploadedBytes), avg(uploadedBytes), max(uploadedBytes) by bin (1m)
```

Kami menyertakan agregat ini karena mungkin lebih berguna untuk memvisualisasikan ukuran file minimum, maksimum, dan rata-rata untuk menemukan outlier. Anda dapat melihat hasilnya di tab Visualisasi:



Setelah Anda selesai membangun visualisasi, Anda dapat menambahkan grafik ke dasbor secara opsional. CloudWatch Untuk melakukan ini, pilih Tambahkan ke dasbor di atas visualisasi. Ini

menambahkan kueri sebagai widget dan Anda dapat memilih interval penyegaran otomatis, sehingga lebih mudah untuk terus memantau hasilnya:



Mengirim log Fungsi Peristiwa Lambda ke Firehose

Konsol Lambda sekarang menawarkan opsi untuk mengirim log Fungsi Acara ke Firehose. Ini memungkinkan streaming real-time log Anda ke berbagai tujuan yang didukung oleh Firehose, termasuk alat analitik pihak ketiga dan titik akhir kustom.

> **Note**
>
> Anda dapat mengonfigurasi log Fungsi Peristiwa Lambda untuk dikirim ke Firehose menggunakan konsol Lambda,,, AWS CLI dan semua SDK. AWS CloudFormation AWS

Harga

Untuk detail tentang harga, lihat CloudWatch harga Amazon.

Izin yang diperlukan untuk tujuan log Firehose

Saat menggunakan konsol Lambda untuk mengonfigurasi Firehose sebagai tujuan log Fungsi Acara, Anda memerlukan:

1. Izin IAM yang diperlukan untuk menggunakan CloudWatch Log dengan Lambda.
2. Untuk mengatur filter berlangganan dengan Firehose. Filter ini menentukan peristiwa log mana yang dikirimkan ke aliran Firehose Anda.

Mengirim log Fungsi Peristiwa Lambda ke Firehose

Di konsol Lambda, Anda dapat mengirim log Fungsi Acara langsung ke Firehose setelah membuat Fungsi Acara baru. Untuk melakukan ini, selesaikan langkah-langkah ini:

1. Masuk ke Konsol AWS Manajemen dan buka konsol Lambda.
2. Pilih nama Event Function Anda.
3. Pilih tab Konfigurasi.
4. Pilih tab Alat Pemantauan dan Operasi.
5. Di bagian “Konfigurasi logging”, pilih Edit.
6. Di bagian “Konten log”, pilih format log.
7. Di bagian “Log destination”, selesaikan langkah-langkah berikut:
   a. Pilih layanan tujuan.
   b. Pilih untuk membuat grup log baru atau menggunakan grup log yang ada.

      > **Note**
      >
      > Jika memilih grup log yang ada untuk tujuan Firehose, pastikan grup log yang Anda pilih adalah jenis grup `Delivery` log.

   c. Pilih aliran Firehose.
   d. Grup CloudWatch `Delivery` log akan muncul.
8. Pilih Simpan.

> **Note**
>
> Jika peran IAM yang disediakan di konsol tidak memiliki izin yang diperlukan, maka pengaturan tujuan gagal. Untuk memperbaikinya, lihat Izin yang diperlukan untuk tujuan log Firehose untuk memberikan izin yang diperlukan.

Cross-Account Penebangan

Anda dapat mengonfigurasi Lambda untuk mengirim log ke aliran pengiriman Firehose di akun lain. AWS Ini memerlukan pengaturan tujuan dan mengonfigurasi izin yang sesuai di kedua akun.

Untuk petunjuk mendetail tentang pengaturan pencatatan lintas akun, termasuk peran dan kebijakan IAM yang diperlukan, lihat Menyiapkan langganan lintas akun baru di dokumentasi Log. CloudWatch

Mengirim log Fungsi Peristiwa Lambda ke Amazon S3

Anda dapat mengonfigurasi Fungsi Peristiwa Lambda Anda untuk mengirim log langsung ke Amazon S3 menggunakan konsol Lambda. Fitur ini memberikan solusi hemat biaya untuk penyimpanan log jangka panjang dan memungkinkan opsi analisis yang kuat menggunakan layanan seperti Athena.

> **Note**
>
> Anda dapat mengonfigurasi log Fungsi Peristiwa Lambda untuk dikirim ke Amazon S3 menggunakan konsol Lambda AWS CLI,,, AWS CloudFormation dan semua SDK. AWS

Harga

Untuk detail tentang harga, lihat CloudWatch harga Amazon.

Izin yang diperlukan untuk tujuan log Amazon S3

Saat menggunakan konsol Lambda untuk mengonfigurasi Amazon S3 sebagai tujuan log Fungsi Acara, Anda memerlukan:

1. Izin IAM yang diperlukan untuk menggunakan CloudWatch Log dengan Lambda.
2. UntukSiapkan filter langganan CloudWatch Log untuk mengirim log Fungsi Peristiwa Lambda ke Amazon S3. Filter ini menentukan peristiwa log mana yang dikirimkan ke bucket Amazon S3 Anda.

Siapkan filter langganan CloudWatch Log untuk mengirim log Fungsi Peristiwa Lambda ke Amazon S3

Untuk mengirim CloudWatch log dari Log ke Amazon S3, Anda perlu membuat filter langganan. Filter ini menentukan peristiwa log mana yang dikirimkan ke bucket Amazon S3 Anda. Bucket Amazon S3 Anda harus berada di Wilayah yang sama dengan grup log Anda.

Untuk membuat filter berlangganan untuk Amazon S3

1. Buat bucket Amazon Simple Storage Service (Amazon S3). Kami menyarankan Anda menggunakan bucket yang dibuat khusus untuk CloudWatch Log. Namun, jika Anda ingin menggunakan bucket yang sudah ada, lewati ke langkah 2.

Jalankan perintah berikut, dengan mengganti Wilayah placeholder dengan Wilayah yang ingin Anda gunakan:

```
aws s3api create-bucket --bucket amzn-s3-demo-bucket2 --create-bucket-configuration
 LocationConstraint=region
```

> **Note**
>
> `amzn-s3-demo-bucket2`adalah contoh nama bucket Amazon S3. Itu dicadangkan. Agar prosedur ini berfungsi, Anda harus menggantinya dengan nama bucket Amazon S3 unik Anda.

Berikut ini adalah output contoh:

```
{
    "Location": "/amzn-s3-demo-bucket2"
}
```

2. Buat peran IAM yang memberikan izin CloudWatch Log untuk memasukkan data ke dalam bucket Amazon S3 Anda. Kebijakan ini mencakup kunci konteks kondisi SourceArn global aws: untuk membantu mencegah masalah keamanan wakil yang membingungkan. Untuk informasi lebih lanjut, lihat Pencegahan wakil yang bingung.

   a. Gunakan editor teks untuk membuat kebijakan kepercayaan dalam file *~/*`TrustPolicyForCWL.json` sebagai berikut:

```
{
    "Statement": {
        "Effect": "Allow",
        "Principal": { "Service": "logs.amazonaws.com" },
        "Condition": {
            "StringLike": {
                "aws:SourceArn": "arn:aws:logs:region:123456789012:*"
            }
         },
        "Action": "sts:AssumeRole"
    }
}
```

b. Gunakan perintah create-role untuk membuat IAM role, dengan menentukan file kebijakan kepercayaan. Catatan Role.Arn nilai yang dikembalikan, karena Anda membutuhkannya di langkah selanjutnya:

```
aws iam create-role \
 --role-name CWLtoS3Role \
 --assume-role-policy-document file://~/TrustPolicyForCWL.json
{
    "Role": {
        "AssumeRolePolicyDocument": {
            "Statement": {
                "Action": "sts:AssumeRole",
                "Effect": "Allow",
                "Principal": {
                    "Service": "logs.amazonaws.com"
                },
                "Condition": {
                    "StringLike": {
                        "aws:SourceArn": "arn:aws:logs:region:123456789012:*"
                    }
                }
            }
        },
        "RoleId": "AAOIIAH450GAB4HC5F431",
        "CreateDate": "2015-05-29T13:46:29.431Z",
        "RoleName": "CWLtoS3Role",
        "Path": "/",
        "Arn": "arn:aws:iam::123456789012:role/CWLtoS3Role"
    }
}
```

3. Buat kebijakan izin untuk menentukan tindakan apa yang dapat dilakukan CloudWatch Log pada akun Anda. Pertama, gunakan editor teks untuk membuat kebijakan izin dalam file `~/PermissionsForCWL.json`:

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["s3:PutObject"],
      "Resource": ["arn:aws:s3:::amzn-s3-demo-bucket2/*"]
    }
```

```
  ]
}
```

Kaitkan kebijakan izin dengan peran menggunakan `put-role-policy` perintah berikut:

```
aws iam put-role-policy --role-name CWLtoS3Role --policy-name Permissions-Policy-For-S3 --policy-document file://~/PermissionsForCWL.json
```

4. Buat grup `Delivery` log atau gunakan grup `Delivery` log yang ada.

```
aws logs create-log-group --log-group-name my-logs --log-group-class DELIVERY --region REGION_NAME
```

5. `PutSubscriptionFilter`untuk mengatur tujuan

```
aws logs put-subscription-filter
--log-group-name my-logs
--filter-name my-lambda-delivery
--filter-pattern ""
--destination-arn arn:aws:s3:::amzn-s3-demo-bucket2
--role-arn arn:aws:iam::123456789012:role/CWLtoS3Role
--region REGION_NAME
```

Mengirim log Fungsi Peristiwa Lambda ke Amazon S3

Di konsol Lambda, Anda dapat mengirim log Fungsi Acara langsung ke Amazon S3 setelah membuat Fungsi Acara baru. Untuk melakukan ini, selesaikan langkah-langkah ini:

1. Masuk ke Konsol AWS Manajemen dan buka konsol Lambda.
2. Pilih nama Event Function Anda.
3. Pilih tab Konfigurasi.
4. Pilih tab Alat Pemantauan dan Operasi.
5. Di bagian “Konfigurasi logging”, pilih Edit.
6. Di bagian “Konten log”, pilih format log.
7. Di bagian “Log destination”, selesaikan langkah-langkah berikut:
   a. Pilih layanan tujuan.
   b. Pilih untuk membuat grup log baru atau menggunakan grup log yang ada.

> **Note**
>
> Jika memilih grup log yang ada untuk tujuan Amazon S3, pastikan grup log yang Anda pilih adalah jenis grup `Delivery` log.

c. Pilih bucket Amazon S3 untuk menjadi tujuan log Fungsi Acara Anda.

d. Grup CloudWatch `Delivery` log muncul.

8. Pilih Simpan.

> **Note**
>
> Jika peran IAM yang disediakan di konsol tidak memiliki izin yang diperlukan, maka pengaturan tujuan gagal. Untuk memperbaikinya, lihat Izin yang diperlukan untuk tujuan log Amazon S3.

Cross-Account Penebangan

Anda dapat mengonfigurasi Lambda untuk mengirim log ke bucket Amazon S3 di akun lain. AWS Ini memerlukan pengaturan tujuan dan mengonfigurasi izin yang sesuai di kedua akun.

Untuk petunjuk mendetail tentang pengaturan pencatatan lintas akun, termasuk peran dan kebijakan IAM yang diperlukan, lihat Menyiapkan langganan lintas akun baru di dokumentasi Log. CloudWatch

Pencatatan log AWS Lambda Panggilan API menggunakan AWS CloudTrail

AWS Lambda terintegrasi dengan AWS CloudTrail, layanan yang menyediakan catatan tindakan yang diambil oleh pengguna, peran, atau Layanan AWS. CloudTrail menangkap panggilan API untuk Lambda sebagai peristiwa. Panggilan yang diambil termasuk panggilan dari konsol Lambda dan panggilan kode ke operasi API Lambda. Dengan menggunakan informasi yang dikumpulkan oleh CloudTrail, Anda dapat menentukan permintaan yang dibuat untuk Lambda, alamat IP dari mana permintaan itu dibuat, kapan dibuat, dan detail tambahan.

Setiap entri peristiwa atau log berisi informasi tentang entitas yang membuat permintaan tersebut. Informasi identitas membantu Anda menentukan berikut hal ini:

- Baik permintaan tersebut dibuat dengan kredensial pengguna root atau pengguna.
- Apakah permintaan dibuat atas nama pengguna IAM Identity Center.
- Apakah permintaan tersebut dibuat dengan kredensial keamanan sementara untuk satu peran atau pengguna gabungan.
- Apakah permintaan tersebut dibuat oleh Layanan AWS lain.

CloudTrail aktif di Anda Akun AWS ketika Anda membuat akun dan Anda secara otomatis memiliki akses ke riwayat CloudTrail Acara. Riwayat CloudTrail Acara menyediakan catatan yang dapat dilihat, dapat dicari, dapat diunduh, dan tidak dapat diubah dari 90 hari terakhir dari peristiwa manajemen yang direkam dalam file. Wilayah AWS Untuk informasi selengkapnya, lihat Bekerja dengan riwayat CloudTrail Acara di Panduan AWS CloudTrail Pengguna. Tidak ada CloudTrail biaya untuk melihat riwayat Acara.

Untuk catatan acara yang sedang berlangsung dalam 90 hari Akun AWS terakhir Anda, buat jejak atau penyimpanan data acara CloudTrailDanau.

CloudTrail jalan setapak

> Jejak memungkinkan CloudTrail untuk mengirimkan file log ke bucket Amazon S3. Semua jalur yang dibuat menggunakan Konsol Manajemen AWS Multi-region. Anda dapat membuat jalur Single-region atau Multi-region dengan menggunakan. AWS CLI Membuat jejak Multi-wilayah disarankan karena Anda menangkap aktivitas Wilayah AWS di semua akun Anda. Jika Anda membuat jejak wilayah Tunggal, Anda hanya dapat melihat peristiwa yang dicatat di jejak. Wilayah AWS Untuk informasi selengkapnya tentang jejak, lihat Membuat jejak untuk Anda Akun AWS dan Membuat jejak untuk organisasi di Panduan AWS CloudTrail Pengguna.

Anda dapat mengirimkan satu salinan acara manajemen yang sedang berlangsung ke bucket Amazon S3 Anda tanpa biaya CloudTrail dengan membuat jejak, namun, ada biaya penyimpanan Amazon S3. Untuk informasi selengkapnya tentang CloudTrail harga, lihat AWS CloudTrail Harga. Untuk informasi tentang harga Amazon S3, lihat Harga Amazon S3.

CloudTrail Menyimpan data acara danau

CloudTrail Lake memungkinkan Anda menjalankan SQL-based kueri pada acara Anda. CloudTrail Lake mengonversi peristiwa yang ada dalam format JSON berbasis baris ke format Apache ORC. ORC adalah format penyimpanan kolumnar yang dioptimalkan untuk pengambilan data dengan cepat. Peristiwa digabungkan ke dalam penyimpanan data peristiwa, yang merupakan kumpulan peristiwa yang tidak dapat diubah berdasarkan kriteria yang Anda pilih dengan menerapkan pemilih acara tingkat lanjut. Penyeleksi yang Anda terapkan ke penyimpanan data acara mengontrol peristiwa mana yang bertahan dan tersedia untuk Anda kueri. Untuk informasi lebih lanjut tentang CloudTrail Danau, lihat Bekerja dengan AWS CloudTrail Danau di Panduan AWS CloudTrail Pengguna.

CloudTrail Penyimpanan data acara danau dan kueri menimbulkan biaya. Saat Anda membuat penyimpanan data acara, Anda memilih opsi harga yang ingin Anda gunakan untuk penyimpanan data acara. Opsi penetapan harga menentukan biaya untuk menelan dan menyimpan peristiwa, dan periode retensi default dan maksimum untuk penyimpanan data acara. Untuk informasi selengkapnya tentang CloudTrail harga, lihat AWS CloudTrail Harga.

Peristiwa data Lambda di CloudTrail

Peristiwa data memberikan informasi tentang operasi sumber daya yang dilakukan pada atau di sumber daya (misalnya, membaca atau menulis ke objek Amazon S3). Ini juga dikenal sebagai operasi bidang data. Peristiwa data sering kali merupakan aktivitas bervolume tinggi. Secara default, CloudTrail tidak mencatat sebagian besar peristiwa data, dan riwayat CloudTrail Peristiwa tidak merekamnya.

Satu peristiwa CloudTrail data yang dicatat secara default untuk layanan yang didukung adalah`LambdaESMDisabled`. Untuk mempelajari lebih lanjut tentang menggunakan acara ini untuk membantu memecahkan masalah dengan pemetaan sumber peristiwa Lambda, lihat. the section called “Menggunakan CloudTrail untuk memecahkan masalah sumber acara Lambda yang dinonaktifkan”

Biaya tambahan berlaku untuk peristiwa data. Untuk informasi selengkapnya tentang CloudTrail harga, lihat AWS CloudTrail Harga.

Anda dapat mencatat peristiwa data untuk jenis `AWS::Lambda::Function` sumber daya menggunakan CloudTrail konsol AWS CLI, atau operasi CloudTrail API. Untuk informasi selengkapnya tentang cara mencatat peristiwa data, lihat Mencatat peristiwa data dengan Konsol Manajemen AWS dan Logging peristiwa data dengan AWS Command Line Interface di Panduan AWS CloudTrail Pengguna.

Tabel berikut mencantumkan jenis sumber daya Lambda yang dapat Anda log peristiwa data. Kolom tipe peristiwa data (konsol) menunjukkan nilai yang akan dipilih dari daftar tipe peristiwa Data di CloudTrail konsol. Kolom nilai resources.type menunjukkan **`resources.type`** nilai, yang akan Anda tentukan saat mengonfigurasi penyeleksi acara lanjutan menggunakan API atau. AWS CLI CloudTrail CloudTrailKolom API Data yang dicatat ke menampilkan panggilan API yang dicatat CloudTrail untuk jenis sumber daya.

Jenis peristiwa data (konsol)	nilai resources.type	API data masuk CloudTrail
Lambda	`AWS::Lambda::Function`	Memohon

Anda dapat mengonfigurasi pemilih acara lanjutan untuk memfilter pada `eventName`,`readOnly`, dan `resources.ARN` bidang untuk mencatat hanya peristiwa yang penting bagi Anda. Contoh berikut adalah tampilan JSON dari konfigurasi peristiwa data yang mencatat peristiwa untuk Fungsi Acara tertentu saja. Untuk informasi selengkapnya tentang bidang ini, lihat AdvancedFieldSelectordi Referensi AWS CloudTrail API.

```
[
  {
    "name": "function-invokes",
    "fieldSelectors": [
      {
        "field": "eventCategory",
        "equals": [
          "Data"
        ]
      },
      {
        "field": "resources.type",
        "equals": [
          "AWS::Lambda::Function"
        ]
```

```
      },
      {
        "field": "resources.ARN",
        "equals": [
          "arn:aws:lambda:us-east-1:111122223333:function:hello-world"
        ]
      }
    ]
  }
]
```

Acara manajemen Lambda di CloudTrail

Acara manajemen memberikan informasi tentang operasi manajemen yang dilakukan pada sumber daya di Anda Akun AWS. Ini juga dikenal sebagai operasi bidang kontrol. Secara default, CloudTrail mencatat peristiwa manajemen.

Lambda mendukung pencatatan tindakan berikut sebagai peristiwa manajemen dalam file CloudTrail log.

> **Note**
>
> Dalam file CloudTrail log, `eventName` mungkin menyertakan informasi tanggal dan versi, tetapi masih mengacu pada tindakan API publik yang sama. Misalnya, `GetFunction` tindakan muncul sebagai`GetFunction20150331v2`. Daftar berikut menentukan kapan nama acara berbeda dari nama tindakan API.

- AddLayerVersionPermission
- AddPermission(nama acara:`AddPermission20150331v2`)
- CreateAlias(nama acara:`CreateAlias20150331`)
- CreateEventSourceMapping(nama acara:`CreateEventSourceMapping20150331`)
- CreateFunction(nama acara:`CreateFunction20150331`)

  (`ZipFile`Parameter `Environment` dan dihilangkan dari CloudTrail log untuk`CreateFunction`.)
- CreateFunctionUrlConfig
- DeleteAlias(nama acara:`DeleteAlias20150331`)
- DeleteCodeSigningConfig

- DeleteEventSourceMapping(nama acara:`DeleteEventSourceMapping20150331`)
- DeleteFunction(nama acara:`DeleteFunction20150331`)
- DeleteFunctionConcurrency(nama acara:`DeleteFunctionConcurrency20171031`)
- DeleteFunctionUrlConfig
- DeleteProvisionedConcurrencyConfig
- GetAlias(nama acara:`GetAlias20150331`)
- GetEventSourceMapping
- GetFunction
- GetFunctionUrlConfig
- GetFunctionConfiguration
- GetLayerVersionPolicy
- GetPolicy
- ListEventSourceMappings
- ListFunctions
- ListFunctionUrlConfigs
- PublishLayerVersion(nama acara:`PublishLayerVersion20181031`)

  (`ZipFile`Parameter dihilangkan dari CloudTrail log untuk`PublishLayerVersion`.)
- PublishVersion(nama acara:`PublishVersion20150331`)
- PutFunctionConcurrency(nama acara:`PutFunctionConcurrency20171031`)
- PutFunctionCodeSigningConfig
- PutFunctionEventInvokeConfig
- PutProvisionedConcurrencyConfig
- PutRuntimeManagementConfig
- RemovePermission(nama acara:`RemovePermission20150331v2`)
- TagResource(nama acara:`TagResource20170331v2`)
- UntagResource(nama acara:`UntagResource20170331v2`)
- UpdateAlias(nama acara:`UpdateAlias20150331`)
- UpdateCodeSigningConfig
- UpdateEventSourceMapping(nama acara:`UpdateEventSourceMapping20150331`)

- UpdateFunctionCode(nama acara:`UpdateFunctionCode20150331v2`)

  (`ZipFile`Parameter dihilangkan dari CloudTrail log untuk`UpdateFunctionCode`.)
- UpdateFunctionConfiguration(nama acara:`UpdateFunctionConfiguration20150331v2`)

  (`Environment`Parameter dihilangkan dari CloudTrail log untuk`UpdateFunctionConfiguration`.)
- UpdateFunctionEventInvokeConfig
- UpdateFunctionUrlConfig

Menggunakan CloudTrail untuk memecahkan masalah sumber acara Lambda yang dinonaktifkan

Saat Anda mengubah status pemetaan sumber peristiwa menggunakan tindakan UpdateEventSourceMappingAPI, panggilan API dicatat sebagai peristiwa manajemen. CloudTrail Pemetaan sumber peristiwa juga dapat bertransisi langsung ke `Disabled` status karena kesalahan.

Untuk layanan berikut, Lambda memublikasikan peristiwa `LambdaESMDisabled` data CloudTrail saat sumber peristiwa Anda bertransisi ke status Dinonaktifkan:

- Amazon Simple Queue Service (Amazon SQS)
- Amazon DynamoDB
- Amazon Kinesis

Lambda tidak mendukung acara ini untuk jenis pemetaan sumber peristiwa lainnya.

Untuk menerima peringatan saat pemetaan sumber peristiwa untuk layanan yang didukung beralih ke `Disabled` status, siapkan alarm di Amazon CloudWatch menggunakan acara tersebut. `LambdaESMDisabled` CloudTrail Untuk mempelajari lebih lanjut tentang mengatur CloudWatch alarm, lihat Membuat CloudWatch alarm untuk CloudTrail acara: contoh.

`serviceEventDetails`Entitas dalam pesan `LambdaESMDisabled` acara berisi salah satu kode kesalahan berikut.

`RESOURCE_NOT_FOUND`

Sumber daya yang ditentukan dalam permintaan tidak ada.

FUNCTION_NOT_FOUND

Fungsi Acara yang dilampirkan ke sumber acara tidak ada.

REGION_NAME_NOT_VALID

Nama Wilayah yang diberikan ke sumber acara atau Fungsi Acara tidak valid.

AUTHORIZATION_ERROR

Izin belum diatur, atau tidak dikonfigurasi dengan benar.

FUNCTION_IN_FAILED_STATE

Kode Fungsi Peristiwa tidak dikompilasi, mengalami pengecualian yang tidak dapat dipulihkan, atau penerapan yang buruk telah terjadi.

Contoh acara Lambda

Peristiwa mewakili permintaan tunggal dari sumber manapun dan mencakup informasi tentang operasi API yang diminta, tanggal dan waktu operasi, parameter permintaan, dan sebagainya. CloudTrail file log bukanlah jejak tumpukan yang diurutkan dari panggilan API publik, sehingga peristiwa tidak muncul dalam urutan tertentu.

Contoh berikut menunjukkan entri CloudTrail log untuk `GetFunction` dan `DeleteFunction` tindakan.

> **Note**
>
> `eventName`Mungkin termasuk informasi tanggal dan versi, seperti`"GetFunction20150331"`, tetapi masih mengacu pada API publik yang sama.

```
{
  "Records": [
    {
      "eventVersion": "1.03",
      "userIdentity": {
        "type": "IAMUser",
        "principalId": "A1B2C3D4E5F6G7EXAMPLE",
        "arn": "arn:aws:iam::111122223333:user/myUserName",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
```

```
        "userName": "myUserName"
      },
      "eventTime": "2015-03-18T19:03:36Z",
      "eventSource": "lambda.amazonaws.com",
      "eventName": "GetFunction",
      "awsRegion": "us-east-1",
      "sourceIPAddress": "127.0.0.1",
      "userAgent": "Python-httplib2/0.8 (gzip)",
      "errorCode": "AccessDenied",
      "errorMessage": "User: arn:aws:iam::111122223333:user/myUserName is not
 authorized to perform: lambda:GetFunction on resource: arn:aws:lambda:us-
west-2:111122223333:function:other-acct-function",
      "requestParameters": null,
      "responseElements": null,
      "requestID": "7aebcd0f-cda1-11e4-aaa2-e356da31e4ff",
      "eventID": "e92a3e85-8ecd-4d23-8074-843aabfe89bf",
      "eventType": "AwsApiCall",
      "recipientAccountId": "111122223333"
    },
    {
      "eventVersion": "1.03",
      "userIdentity": {
        "type": "IAMUser",
        "principalId": "A1B2C3D4E5F6G7EXAMPLE",
        "arn": "arn:aws:iam::111122223333:user/myUserName",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "myUserName"
      },
      "eventTime": "2015-03-18T19:04:42Z",
      "eventSource": "lambda.amazonaws.com",
      "eventName": "DeleteFunction20150331",
      "awsRegion": "us-east-1",
      "sourceIPAddress": "127.0.0.1",
      "userAgent": "Python-httplib2/0.8 (gzip)",
      "requestParameters": {
        "functionName": "basic-node-task"
      },
      "responseElements": null,
      "requestID": "a2198ecc-cda1-11e4-aaa2-e356da31e4ff",
      "eventID": "20b84ce5-730f-482e-b2b2-e8fcc87ceb22",
      "eventType": "AwsApiCall",
      "recipientAccountId": "111122223333"
    }
```

```
  ]
}
```

Untuk informasi tentang konten CloudTrail rekaman, lihat konten CloudTrail rekaman di Panduan AWS CloudTrail Pengguna.

Visualisasikan pemanggilan Fungsi Acara Lambda menggunakan AWS X-Ray

Anda dapat menggunakan AWS X-Ray untuk memvisualisasikan komponen aplikasi Anda, mengidentifikasi kemacetan kinerja, dan memecahkan masalah permintaan yang mengakibatkan kesalahan. Fungsi Peristiwa Lambda Anda mengirim data jejak ke X-Ray, dan X-Ray memproses data untuk menghasilkan peta layanan dan ringkasan jejak yang dapat dicari.

Lambda mendukung dua mode penelusuran untuk X-Ray: dan. `Active` `PassThrough` Dengan `Active` penelusuran, Lambda secara otomatis membuat segmen jejak untuk pemanggilan Fungsi Acara dan mengirimkannya ke. X-Ray `PassThrough`mode, di sisi lain, hanya menyebarkan konteks penelusuran ke layanan hilir.

Jika Anda telah mengaktifkan `Active` penelusuran untuk Fungsi Acara, Lambda secara otomatis mengirimkan jejak X-Ray untuk permintaan sampel. Biasanya, layanan upstream, seperti Amazon API Gateway atau aplikasi yang dihosting di Amazon EC2 yang diinstrumentasi dengan SDK, memutuskan apakah permintaan X-Ray yang masuk harus dilacak, lalu menambahkan keputusan pengambilan sampel tersebut sebagai header penelusuran. Lambda menggunakan header itu untuk memutuskan mengirim jejak atau tidak. Jejak dari produsen pesan hulu, seperti Amazon SQS, secara otomatis ditautkan ke jejak dari Fungsi Acara Lambda hilir, menciptakan tampilan ujung ke ujung dari seluruh aplikasi. Untuk informasi selengkapnya, lihat Menelusuri aplikasi berbasis peristiwa di Panduan Pengembang.AWS X-Ray

> **Note**
>
> X-Ray penelusuran saat ini tidak didukung untuk Lambda Event Functions dengan Amazon Managed Streaming for Apache Kafka (Amazon MSK), Apache Kafka yang dikelola sendiri, Amazon MQ dengan ActiveMQ dan RabbitMQ, atau pemetaan sumber acara Amazon DocumentDB.

Untuk mengaktifkan penelusuran aktif di Fungsi Peristiwa Lambda Anda dengan konsol, ikuti langkah-langkah berikut:

Untuk mengaktifkan penelusuran aktif

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih Fungsi Acara.

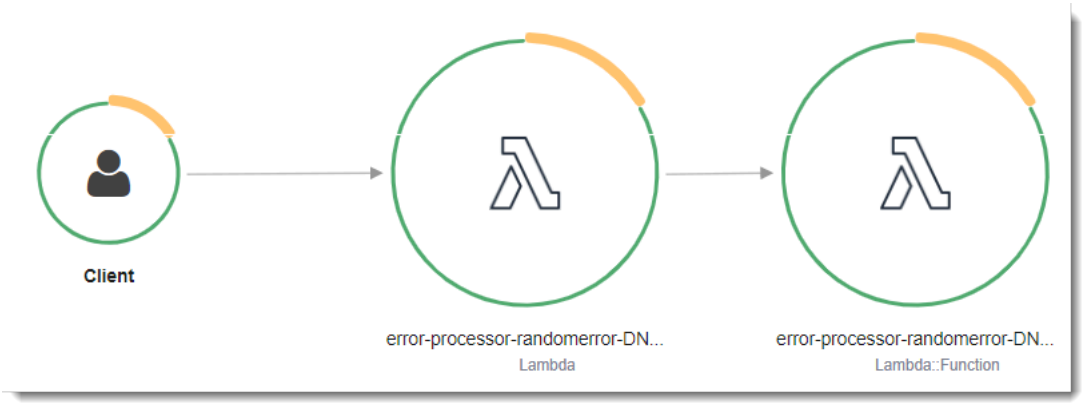
3. Pilih Konfigurasi dan kemudian pilih Alat Pemantauan dan operasi.
4. Di bawah Alat pemantauan tambahan, pilih Edit.
5. Di bawah Sinyal CloudWatch Aplikasi dan AWS X-Ray, pilih Aktifkan untuk jejak layanan Lambda.
6. Pilih Simpan.

Fungsi Acara Anda memerlukan izin untuk mengunggah data jejak ke X-Ray. Saat Anda mengaktifkan penelusuran di konsol Lambda, Lambda menambahkan izin yang diperlukan ke peran eksekusi Fungsi Acara Anda. Atau, tambahkan kebijakan AWSXRayDaemonWriteAccess ke peran eksekusi.

X-Ray tidak melacak semua permintaan ke aplikasi Anda. X-Ray menerapkan algoritma pengambilan sampel untuk memastikan bahwa penelusuran efisien, sambil tetap memberikan sampel yang representatif dari semua permintaan. Tingkat pengambilan sampel adalah 1 permintaan per detik dan 5 persen dari permintaan tambahan. Anda tidak dapat mengonfigurasi laju X-Ray pengambilan sampel untuk fungsi Anda.

Memahami X-Ray jejak

Di X-Ray, jejak mencatat informasi tentang permintaan yang diproses oleh satu atau beberapa layanan. Lambda mencatat 2 segmen per jejak, yang menciptakan dua node pada grafik layanan. Gambar berikut menyoroti dua node ini:



Node pertama di sebelah kiri mewakili layanan Lambda, yang menerima permintaan pemanggilan. Node kedua mewakili Fungsi Peristiwa Lambda spesifik Anda.

Segmen yang direkam untuk layanan Lambda,`AWS::Lambda`, mencakup semua langkah yang diperlukan untuk mempersiapkan lingkungan eksekusi Lambda. Ini termasuk penjadwalan

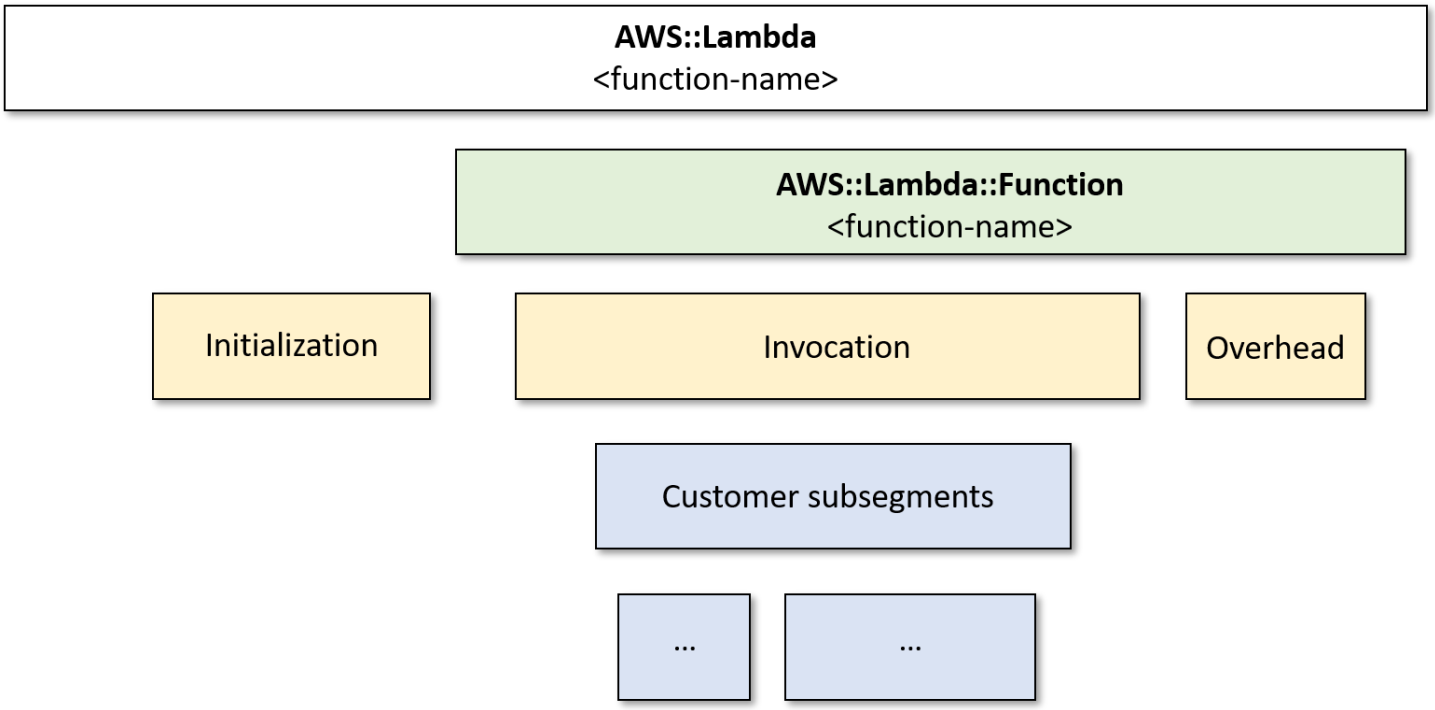
microVM, membuat atau mencairkan lingkungan eksekusi dengan sumber daya yang telah Anda konfigurasikan, serta mengunduh kode fungsi Anda dan semua lapisan.

`AWS::Lambda::Function`Segmen ini untuk pekerjaan yang dilakukan oleh Fungsi Acara.

> **Note**
>
> AWS saat ini menerapkan perubahan pada layanan Lambda. Karena perubahan ini, Anda mungkin melihat perbedaan kecil antara struktur dan konten pesan log sistem dan segmen pelacakan yang dipancarkan oleh Fungsi Peristiwa Lambda yang berbeda di Anda. Akun AWS
> Perubahan ini mempengaruhi subsegmen segmen fungsi. Paragraf berikut menjelaskan format lama dan baru untuk subsegmen ini.
> Perubahan ini akan diterapkan selama beberapa minggu mendatang, dan semua fungsi di semua Wilayah AWS kecuali China GovCloud dan wilayah akan bertransisi untuk menggunakan pesan log format baru dan segmen pelacakan.

Old-style AWS X-Ray Struktur segmen Lambda

X-Ray Struktur gaya lama untuk `AWS::Lambda` segmen ini terlihat seperti berikut:



Dalam format ini, segmen fungsi memiliki subsegmen untuk`Initialization`,`Invocation`, dan`Overhead`. SnapStartHanya saja, ada juga `Restore` subsegmen (tidak ditampilkan pada diagram ini).

Subsegmen `Initialization` merupakan fase init dari siklus hidup lingkungan eksekusi Lambda. Selama fase ini, Lambda menginisialisasi ekstensi, menginisialisasi runtime, dan menjalankan kode inisialisasi fungsi.

Subsegmen `Invocation` merupakan fase invokasi tempat Lambda memanggil handler fungsi. Ini dimulai dengan pendaftaran runtime dan ekstensi serta berakhir ketika runtime siap untuk mengirim respon.

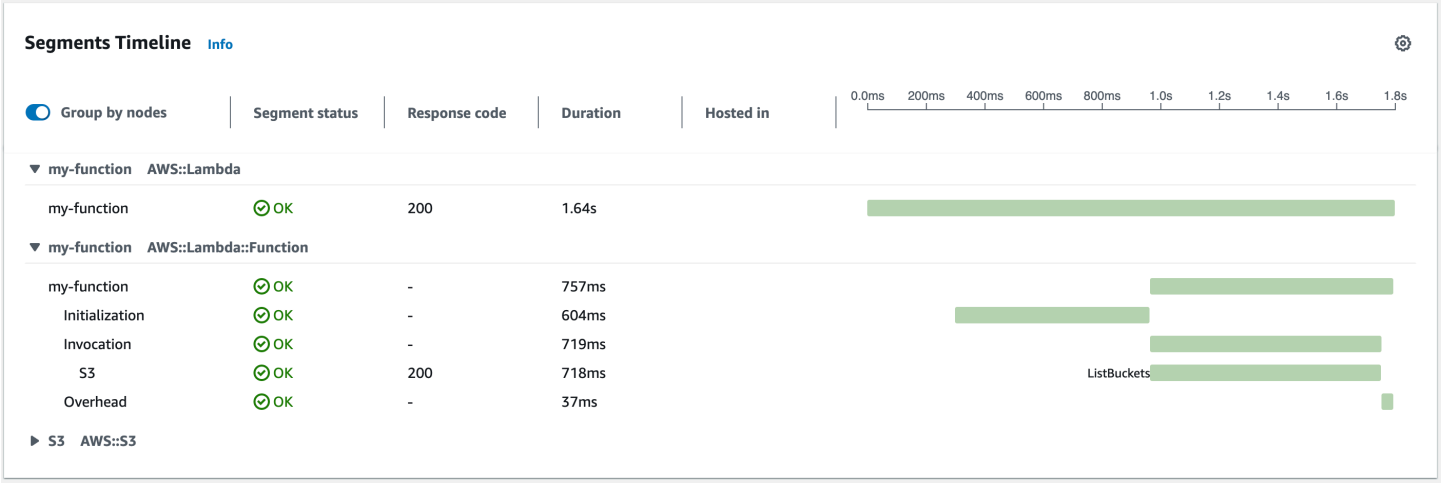
(SnapStart Hanya Lambda) `Restore` Subsegmen menunjukkan waktu yang dibutuhkan Lambda untuk memulihkan snapshot, memuat runtime, dan menjalankan kait runtime setelah pemulihan. Proses memulihkan snapshot dapat mencakup waktu yang dihabiskan untuk aktivitas di luar microVM. Kali ini dilaporkan di `Restore` subsegmen. Anda tidak dikenakan biaya untuk waktu yang dihabiskan di luar microVM untuk memulihkan snapshot.

Subsegmen `Overhead` merupakan fase yang terjadi antara waktu ketika runtime mengirimkan respon dan sinyal untuk invokasi berikutnya. Selama waktu ini, runtime menyelesaikan semua tugas yang terkait dengan invokasi dan mempersiapkan untuk membekukan sandbox.

> **Important**
>
> Anda dapat menggunakan X-Ray SDK untuk memperluas `Invocation` subsegmen dengan subsegmen tambahan untuk panggilan hilir, anotasi, dan metadata. Anda tidak dapat mengakses segmen fungsi secara langsung atau merekam pekerjaan yang dilakukan di luar lingkup invokasi handler.

Untuk informasi selengkapnya tentang fase lingkungan eksekusi Lambda, lihat. the section called “Lingkungan eksekusi”

Contoh jejak menggunakan X-Ray struktur gaya lama ditunjukkan pada diagram berikut.

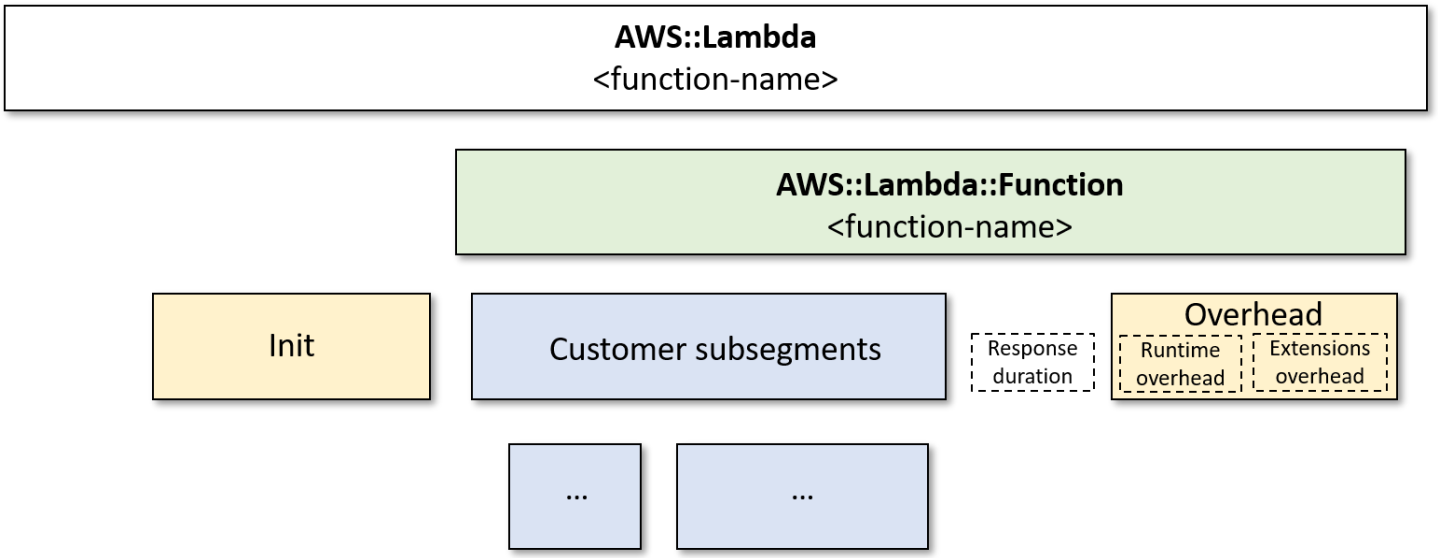


Perhatikan dua segmen dalam contoh. Keduanya bernama fungsi saya, tetapi yang satu memiliki asal `AWS::Lambda` dan yang lainnya memiliki asal usul. `AWS::Lambda::Function` Jika `AWS::Lambda` segmen menunjukkan kesalahan, layanan Lambda mengalami masalah. Jika `AWS::Lambda::Function` segmen menunjukkan kesalahan, Fungsi Acara Anda mengalami masalah.

> **Note**
>
> Kadang-kadang, Anda mungkin melihat kesenjangan besar antara inisialisasi fungsi dan fase pemanggilan dalam jejak Anda. X-Ray Untuk fungsi yang menggunakan konkurensi yang disediakan, ini karena Lambda menginisialisasi instance fungsi Anda jauh sebelum pemanggilan. Untuk Fungsi Peristiwa yang menggunakan konkurensi tanpa syarat (sesuai permintaan), Lambda mungkin secara proaktif menginisialisasi instance fungsi, meskipun tidak ada pemanggilan. Secara visual, kedua kasus ini muncul sebagai celah waktu antara fase inisialisasi dan pemanggilan.

New-style AWS X-Ray Struktur segmen Lambda

X-Ray Struktur gaya baru untuk `AWS::Lambda` segmen ini terlihat seperti berikut:

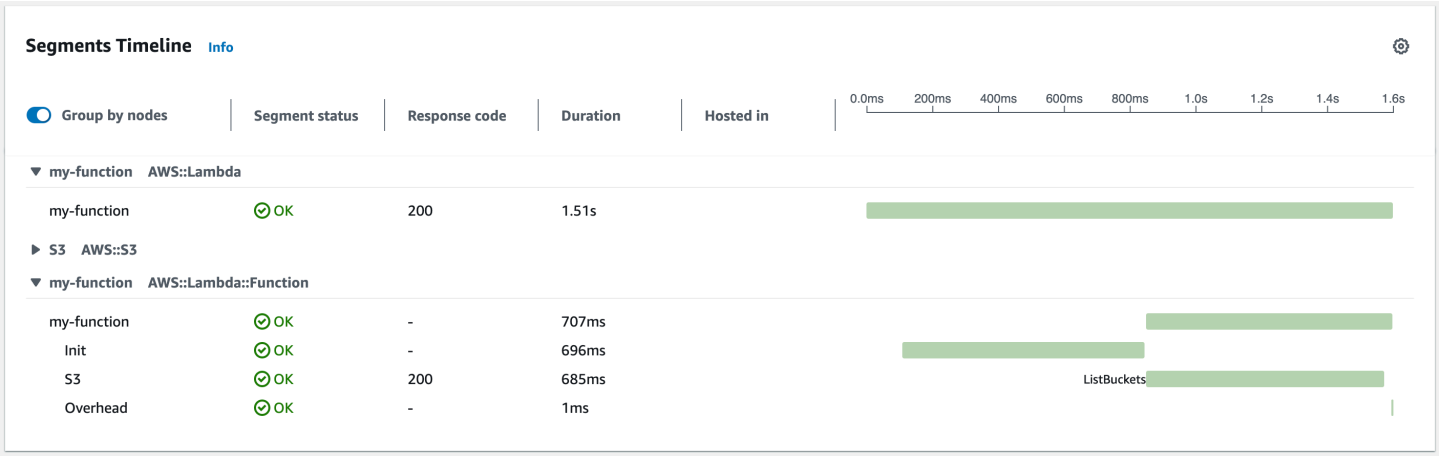


Dalam format baru ini, `Init` Subsegmen mewakili fase init dari siklus hidup lingkungan eksekusi Lambda seperti sebelumnya.

Tidak ada segmen pemanggilan dalam format baru. Sebaliknya, subsegmen pelanggan dilampirkan langsung ke `AWS::Lambda::Function` segmen. Segmen ini berisi metrik berikut sebagai anotasi:

- `aws.responseLatency`- waktu yang dibutuhkan untuk menjalankan fungsi
- `aws.responseDuration`- waktu yang dibutuhkan untuk mentransfer respons kepada pelanggan
- `aws.runtimeOverhead`- jumlah waktu tambahan runtime yang dibutuhkan untuk menyelesaikan
- `aws.extensionOverhead`- jumlah waktu tambahan ekstensi yang dibutuhkan untuk menyelesaikan

Contoh jejak menggunakan X-Ray struktur gaya baru ditunjukkan pada diagram berikut.



Perhatikan dua segmen dalam contoh. Keduanya bernama fungsi saya, tetapi yang satu memiliki asal `AWS::Lambda` dan yang lainnya memiliki asal usul. `AWS::Lambda::Function` Jika `AWS::Lambda` segmen menunjukkan kesalahan, layanan Lambda mengalami masalah. Jika `AWS::Lambda::Function` segmen menunjukkan kesalahan, Fungsi Acara Anda mengalami masalah.

Lihat topik berikut untuk pengantar khusus bahasa tertentu terkait pelacakan di Lambda:

- Menginstrumentasi Node.js kode di AWS Lambda
- Menginstrumentasi kode Python di AWS Lambda
- Menginstrumentasi kode Ruby di AWS Lambda
- Menginstrumentasi kode Java di AWS Lambda
- Menginstrumentasi kode Go di AWS Lambda
- Menginstrumentasi kode C # di AWS Lambda

Untuk daftar lengkap layanan yang mendukung instrumentasi aktif, lihat Didukung Layanan AWS dalam Panduan AWS X-Ray Pengembang.

Perilaku penelusuran default di Lambda

Jika Anda tidak mengaktifkan `Active` penelusuran, Lambda default `PassThrough` ke mode penelusuran.

Dalam `PassThrough` mode, Lambda meneruskan header X-Ray penelusuran ke layanan hilir, tetapi tidak mengirim jejak secara otomatis. Ini benar bahkan jika header tracing berisi keputusan untuk mengambil sampel permintaan. Jika layanan upstream tidak menyediakan header X-Ray tracing, Lambda menghasilkan header dan membuat keputusan untuk tidak mengambil sampel. Namun, Anda dapat mengirim jejak Anda sendiri dengan memanggil pustaka penelusuran dari kode Fungsi Acara Anda.

> **Note**
>
> Sebelumnya, Lambda akan mengirim jejak secara otomatis ketika layanan upstream, seperti Amazon API Gateway, menambahkan header penelusuran. Dengan tidak mengirimkan jejak secara otomatis, Lambda memberi Anda kontrol untuk melacak Fungsi Acara yang penting bagi Anda. Jika solusi Anda bergantung pada perilaku penelusuran pasif ini, beralihlah ke `Active` penelusuran.

Izin peran eksekusi

Lambda memerlukan izin berikut untuk mengirim data jejak ke. X-Ray Tambahkan mereka ke peran eksekusi Event Function Anda.

- xray: PutTraceSegments
- xray: PutTelemetryRecords

Izin ini disertakan dalam kebijakan terkelola AWSXRayDaemonWriteAccess.

Mengaktifkan penelusuran **`Aktif`** dengan API Lambda

Untuk mengelola konfigurasi penelusuran dengan AWS CLI atau AWS SDK, gunakan operasi API berikut:

- UpdateFunctionConfiguration
- GetFunctionConfiguration
- CreateFunction

Contoh AWS CLI perintah berikut memungkinkan penelusuran aktif pada fungsi bernama my-function.

```
aws lambda update-function-configuration --function-name my-function \
--tracing-config Mode=Active
```

Mode penelusuran adalah bagian dari konfigurasi khusus versi saat Anda memublikasikan versi fungsi Anda. Anda tidak dapat mengubah mode pelacakan pada versi yang telah diterbitkan.

Mengaktifkan penelusuran **`Aktif`** dengan CloudFormation

Untuk mengaktifkan penelusuran pada `AWS::Lambda::Function` sumber daya dalam CloudFormation templat, gunakan `TracingConfig` properti.

Example fungsi-inline.yml. — Konfigurasi penelusuran

```
Resources:
  function:
    Type: AWS::Lambda::Function
    Properties:
```

```
      TracingConfig:
        Mode: Active
      ...
```

Untuk sumber `AWS::Serverless::Function` daya AWS Serverless Application Model (AWS SAM), gunakan `Tracing` properti.

Example template.yml — Konfigurasi penelusuran

```
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      Tracing: Active
      ...
```

Pantau kinerja Fungsi Acara dengan Amazon CloudWatch Lambda Insights

Amazon CloudWatch Lambda Insights mengumpulkan dan menggabungkan metrik kinerja runtime Fungsi Peristiwa Lambda dan log untuk aplikasi tanpa server Anda. Halaman ini menjelaskan cara mengaktifkan dan menggunakan Lambda Insights untuk mendiagnosis masalah dengan Fungsi Peristiwa Lambda Anda.

Bagian-bagian

- Cara kerja Lambda Insights memantau aplikasi nirserver
- Harga
- Runtime yang didukung
- Mengaktifkan Lambda Insights di konsol Lambda
- Mengaktifkan Lambda Insights secara terprogram
- Menggunakan dasbor Lambda Insights
- Contoh alur kerja untuk mendeteksi anomali Fungsi Peristiwa
- Contoh alur kerja menggunakan kueri untuk memecahkan masalah Fungsi Peristiwa
- Apa selanjutnya?

Cara kerja Lambda Insights memantau aplikasi nirserver

CloudWatch Lambda Insights adalah solusi pemantauan dan pemecahan masalah untuk aplikasi tanpa server yang berjalan. AWS Lambda Solusi ini mengumpulkan, menggabungkan, dan merangkum metrik tingkat sistem termasuk waktu CPU, memori, cakram, dan penggunaan jaringan. Ini juga mengumpulkan, mengumpulkan, dan merangkum informasi diagnostik seperti start dingin dan penutupan pekerja Lambda untuk membantu Anda mengisolasi masalah dengan Fungsi Acara Lambda Anda dan menyelesaikannya dengan cepat.

Lambda Insights menggunakan ekstensi CloudWatch Lambda Insights baru, yang disediakan sebagai lapisan Lambda. Saat Anda mengaktifkan ekstensi ini pada Fungsi Peristiwa Lambda untuk runtime yang didukung, ekstensi ini mengumpulkan metrik tingkat sistem dan memancarkan peristiwa log kinerja tunggal untuk setiap pemanggilan Fungsi Peristiwa Lambda tersebut. CloudWatch menggunakan format metrik tertanam untuk mengekstrak metrik dari peristiwa log. Untuk informasi selengkapnya, lihat Menggunakan AWS Lambda ekstensi.

Lapisan Lambda Insights memperluas `CreateLogStream` dan `PutLogEvents` untuk grup `/aws/lambda-insights/` log.

Harga

Saat Anda mengaktifkan Wawasan Lambda untuk Fungsi Peristiwa Lambda Anda, Lambda Insights melaporkan 8 metrik per Fungsi Peristiwa dan setiap pemanggilan Fungsi Peristiwa mengirimkan sekitar 1KB data log ke. CloudWatch Anda hanya membayar metrik dan log yang dilaporkan untuk Fungsi Acara menurut Lambda Insights. Tidak ada biaya minimum atau kebijakan penggunaan layanan wajib. Anda tidak membayar Lambda Insights jika Fungsi Acara tidak dipanggil. Untuk contoh harga, lihat CloudWatch Harga Amazon.

Runtime yang didukung

Anda dapat menggunakan Lambda Insights dengan runtime mana pun yang mendukung ekstensi Lambda.

Mengaktifkan Lambda Insights di konsol Lambda

Anda dapat mengaktifkan pemantauan Lambda Insights yang ditingkatkan pada Fungsi Acara Lambda yang baru dan yang sudah ada. Saat Anda mengaktifkan Wawasan Lambda tentang Fungsi Peristiwa di konsol Lambda untuk waktu proses yang didukung, Lambda menambahkan ekstensi Wawasan Lambda sebagai lapisan ke Fungsi Peristiwa, dan memverifikasi atau mencoba melampirkan kebijakan ke peran eksekusi Fungsi Acara Anda. CloudWatchLambdaInsightsExecutionRolePolicy

Untuk mengaktifkan Lambda Insights di konsol Lambda

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih Fungsi Acara Anda.
3. Pilih tab Konfigurasi.
4. Di menu sebelah kiri, pilih Alat pemantauan dan operasi.
5. Pada panel Alat pemantauan tambahan, pilih Edit.
6. Di bagian CloudWatch Lambda Insights, aktifkan Pemantauan yang ditingkatkan.
7. Pilih Simpan.

Mengaktifkan Lambda Insights secara terprogram

Anda juga dapat mengaktifkan Lambda Insights menggunakan CLI AWS Command Line Interface (AWS CLI), AWS Serverless Application Model (SAM),, atau. CloudFormation

AWS Cloud Development Kit (AWS CDK)Saat Anda mengaktifkan Lambda Insights secara terprogram pada Fungsi Peristiwa untuk waktu proses yang didukung, CloudWatch lampirkan CloudWatchLambdaInsightsExecutionRolePolicykebijakan tersebut ke peran eksekusi Fungsi Peristiwa Anda.

Untuk informasi selengkapnya, lihat Memulai Wawasan Lambda di Panduan Pengguna Amazon CloudWatch .

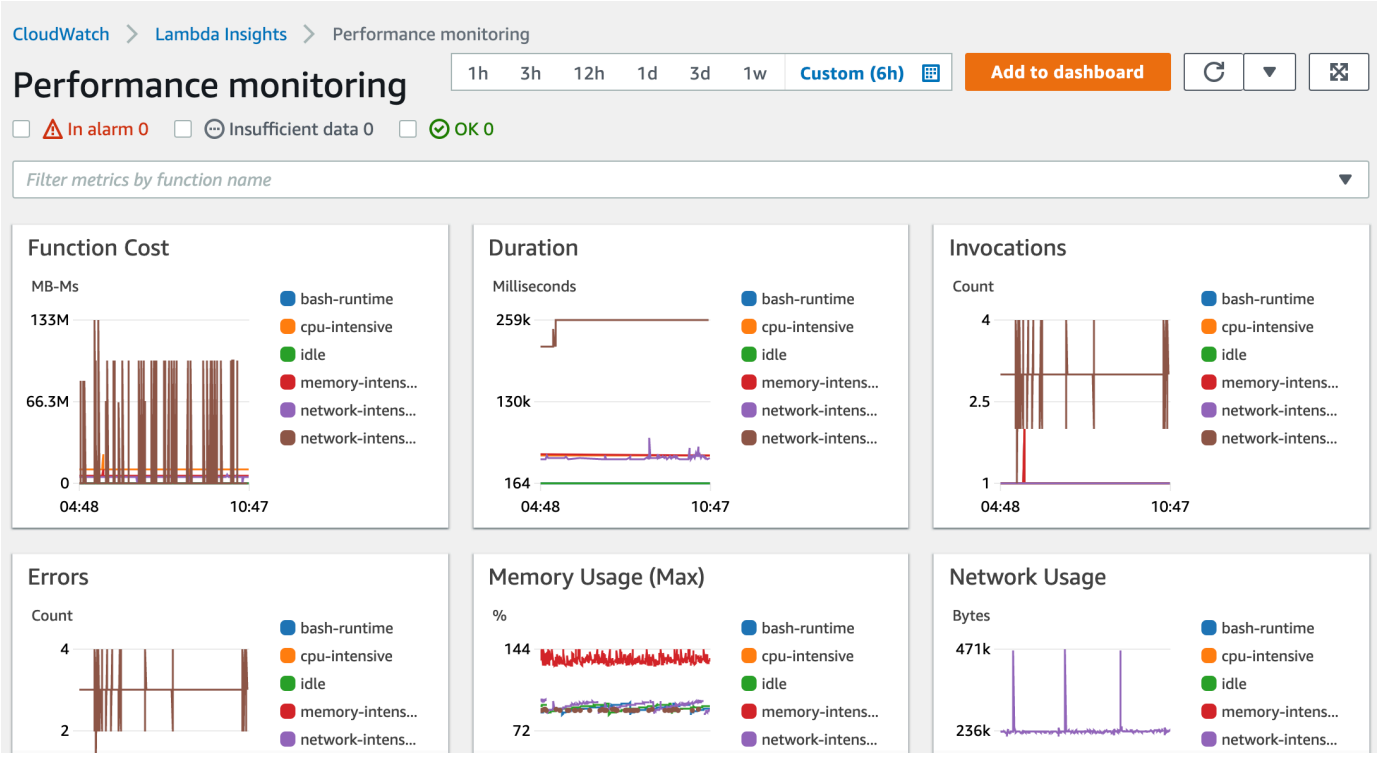
Menggunakan dasbor Lambda Insights

Dasbor Lambda Insights memiliki dua tampilan di CloudWatch konsol: ikhtisar multi-fungsi dan tampilan fungsi tunggal. Ikhtisar multi-fungsi menggabungkan metrik runtime untuk Fungsi Peristiwa Lambda di akun saat ini dan Wilayah. AWS Tampilan fungsi tunggal menunjukkan metrik runtime yang tersedia untuk satu Fungsi Peristiwa Lambda.

Anda dapat menggunakan ikhtisar multi-fungsi dasbor Lambda Insights di CloudWatch konsol untuk mengidentifikasi Fungsi Peristiwa Lambda yang terlalu banyak dan kurang dimanfaatkan. Anda dapat menggunakan tampilan fungsi tunggal dasbor Lambda Insights di CloudWatch konsol untuk memecahkan masalah permintaan individual.

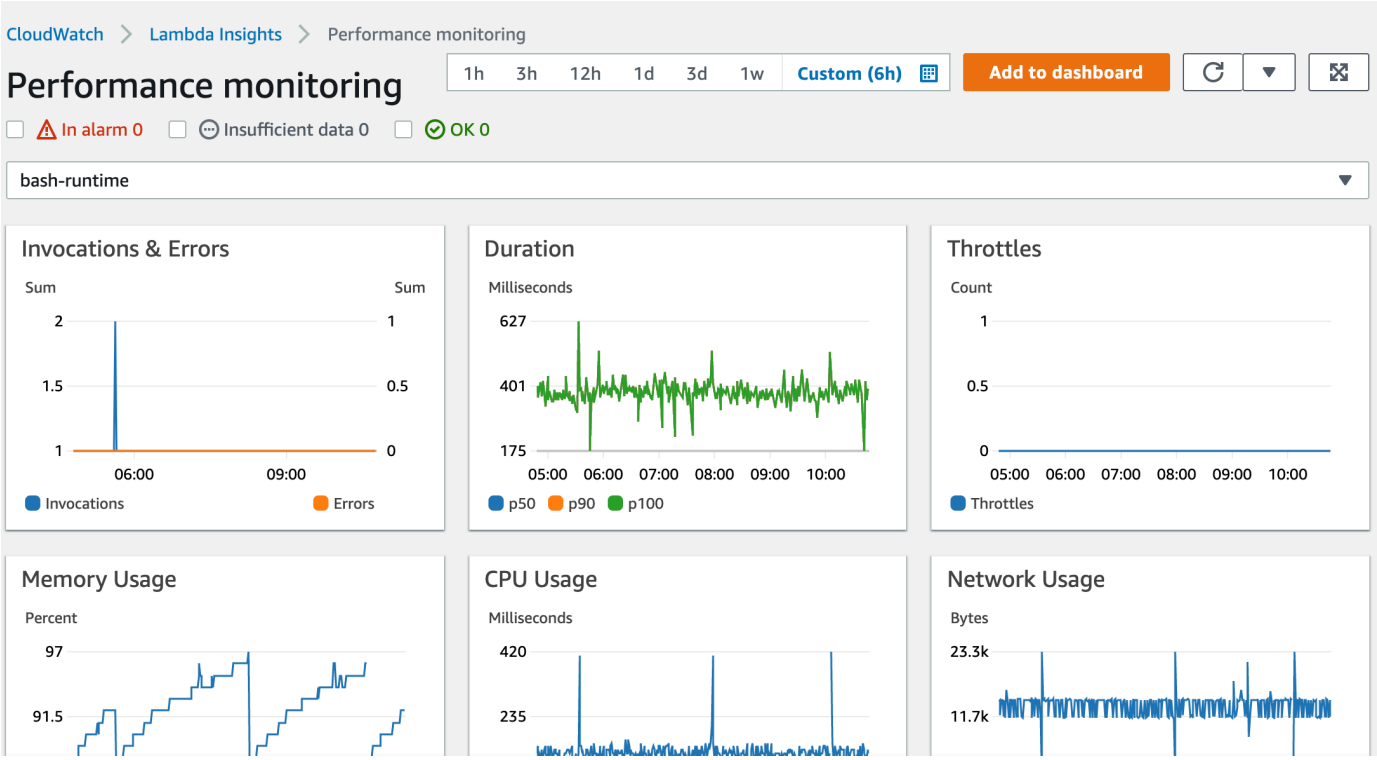
Untuk melihat metrik runtime untuk semua Fungsi Acara

1. Buka Multi-functionhalaman di CloudWatch konsol.
2. Pilih dari rentang waktu yang sudah ditentukan, atau pilih rentang waktu kustom.
3. (Opsional) Pilih Tambahkan ke dasbor untuk menambahkan widget ke CloudWatch dasbor Anda.



Untuk melihat metrik runtime dari satu Fungsi Peristiwa

1. Buka Single-functionhalaman di CloudWatch konsol.
2. Pilih dari rentang waktu yang sudah ditentukan, atau pilih rentang waktu kustom.
3. (Opsional) Pilih Tambahkan ke dasbor untuk menambahkan widget ke CloudWatch dasbor Anda.



Untuk informasi selengkapnya, lihat Membuat dan bekerja dengan widget di CloudWatch dasbor.

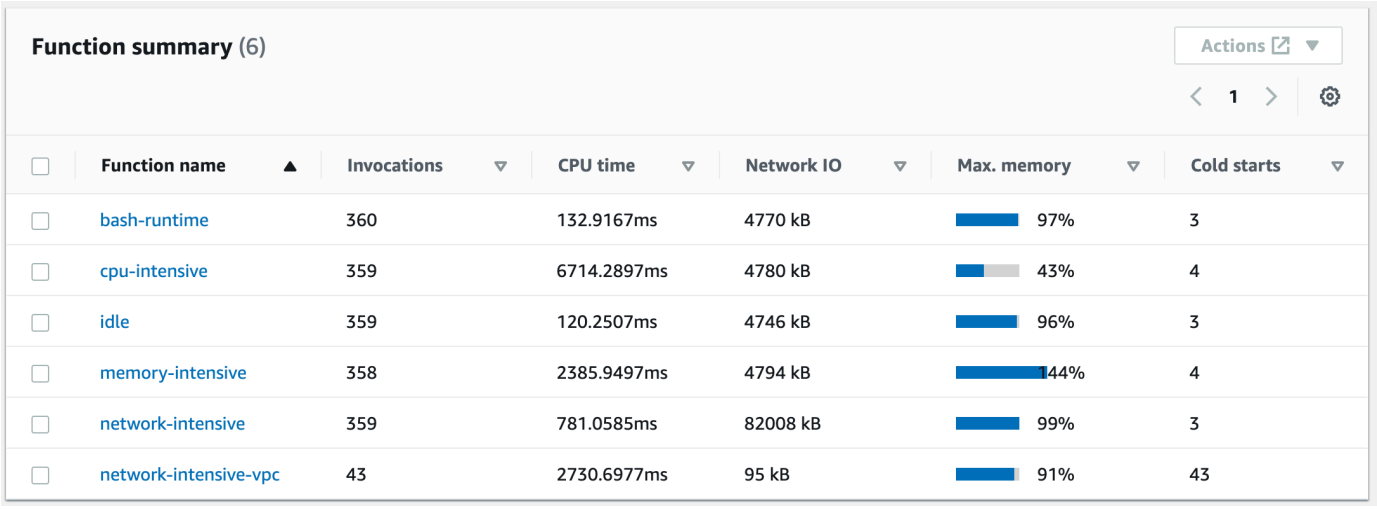
Contoh alur kerja untuk mendeteksi anomali Fungsi Peristiwa

Anda dapat menggunakan ikhtisar multi-fungsi di dasbor Lambda Insights untuk mengidentifikasi dan mendeteksi anomali memori komputasi dengan Fungsi Acara Anda. Misalnya, jika ikhtisar multi-fungsi menunjukkan bahwa Fungsi Peristiwa menggunakan sejumlah besar memori, Anda dapat melihat metrik pemanfaatan memori terperinci di panel Penggunaan Memori. Kemudian, Anda dapat menuju dasbor Metrik untuk mengaktifkan deteksi anomali atau membuat alarm.

Untuk mengaktifkan deteksi anomali untuk Fungsi Peristiwa

1. Buka Multi-functionhalaman di CloudWatch konsol.
2. Di bawah Ringkasan fungsi, pilih nama Fungsi Acara Anda.

   Tampilan fungsi tunggal terbuka dengan metrik runtime Fungsi Acara.



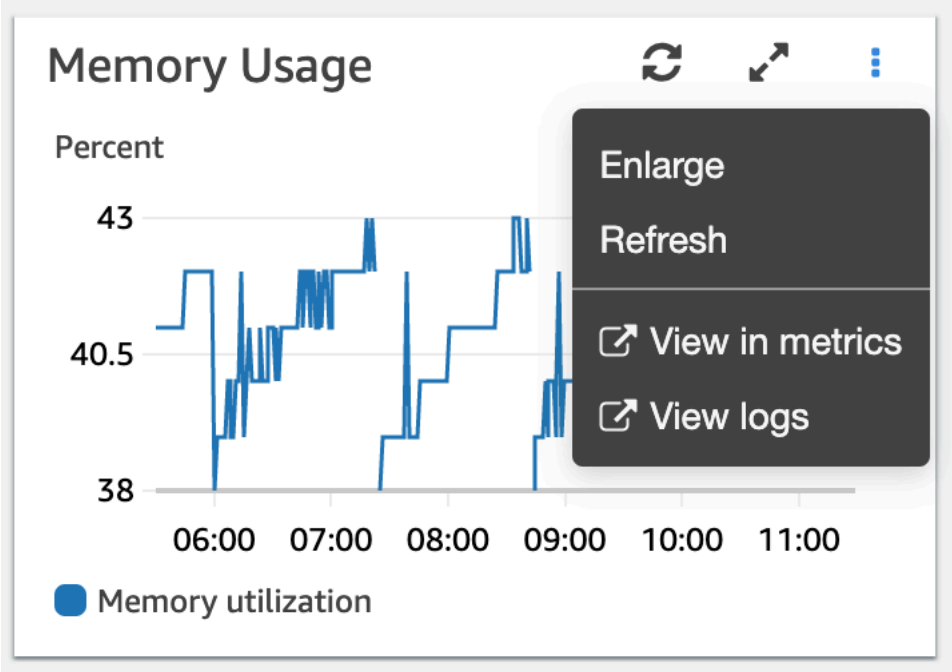
Function summary (6)

Actions

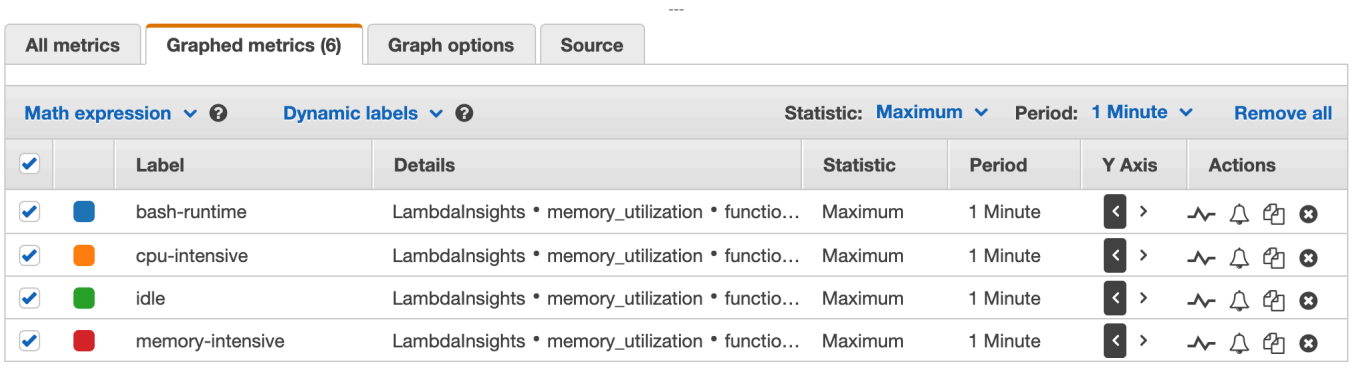
1

Function name	Invocations	CPU time	Network IO	Max. memory	Cold starts
bash-runtime	360	132.9167ms	4770 kB	97%	3
cpu-intensive	359	6714.2897ms	4780 kB	43%	4
idle	359	120.2507ms	4746 kB	96%	3
memory-intensive	358	2385.9497ms	4794 kB	144%	4
network-intensive	359	781.0585ms	82008 kB	99%	3
network-intensive-vpc	43	2730.6977ms	95 kB	91%	43

3. Di panel Penggunaan Memori, pilih tiga titik vertikal, lalu pilih Lihat dalam metrik untuk membuka dasbor Metrik.



4. Pada tab Metrik grafik, di kolom Tindakan, pilih ikon pertama untuk mengaktifkan deteksi anomali untuk Fungsi Peristiwa.



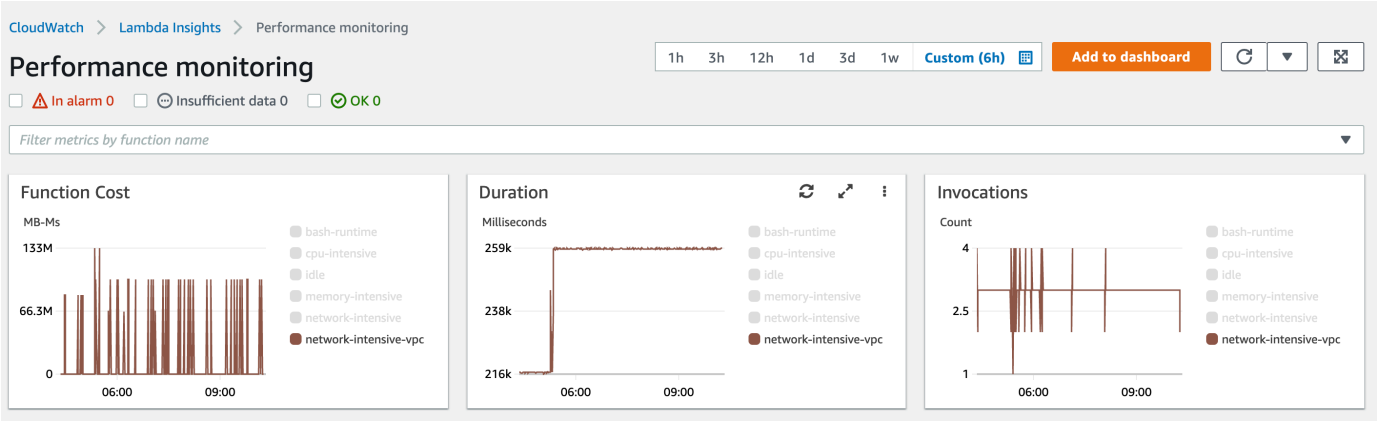
Untuk informasi lebih lanjut, lihat Menggunakan Deteksi CloudWatch Anomali.

Contoh alur kerja menggunakan kueri untuk memecahkan masalah Fungsi Peristiwa

Anda dapat menggunakan tampilan fungsi tunggal di dasbor Lambda Insights untuk mengidentifikasi akar penyebab lonjakan durasi Fungsi Peristiwa. Misalnya, jika ikhtisar multi-fungsi menunjukkan peningkatan besar dalam durasi Fungsi Peristiwa, Anda dapat menjeda atau memilih setiap Fungsi Peristiwa di panel Durasi untuk menentukan Fungsi Peristiwa mana yang menyebabkan peningkatan. Kemudian, Anda dapat masuk ke tampilan fungsi tunggal dan meninjau Log aplikasi untuk menentukan akar masalah.

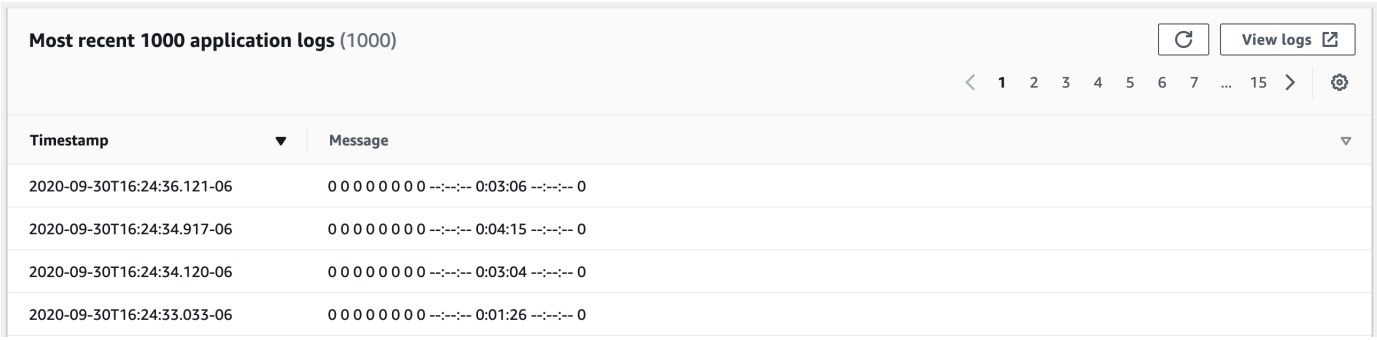
Untuk menjalankan kueri pada Fungsi Acara

1. Buka Multi-functionhalaman di CloudWatch konsol.
2. Di panel Durasi, pilih fungsi Anda untuk memfilter metrik durasi.

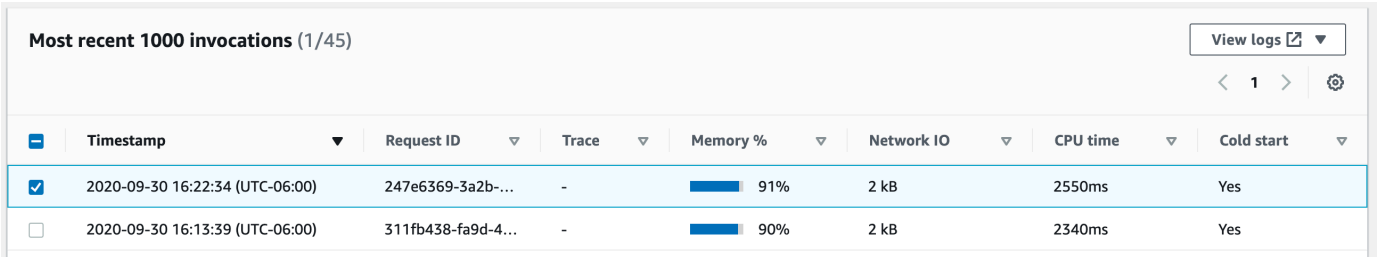


3. Buka Single-functionhalaman.
4. Pilih daftar pilihan menurun Filter metrik berdasarkan nama fungsi, lalu pilih fungsi Anda.
5. Untuk melihat 1000 log aplikasi terbaru, pilih tab Log aplikasi.

6. Meninjau Stempel waktu dan Pesan untuk mengidentifikasi permintaan invokasi yang ingin Anda pecahkan.



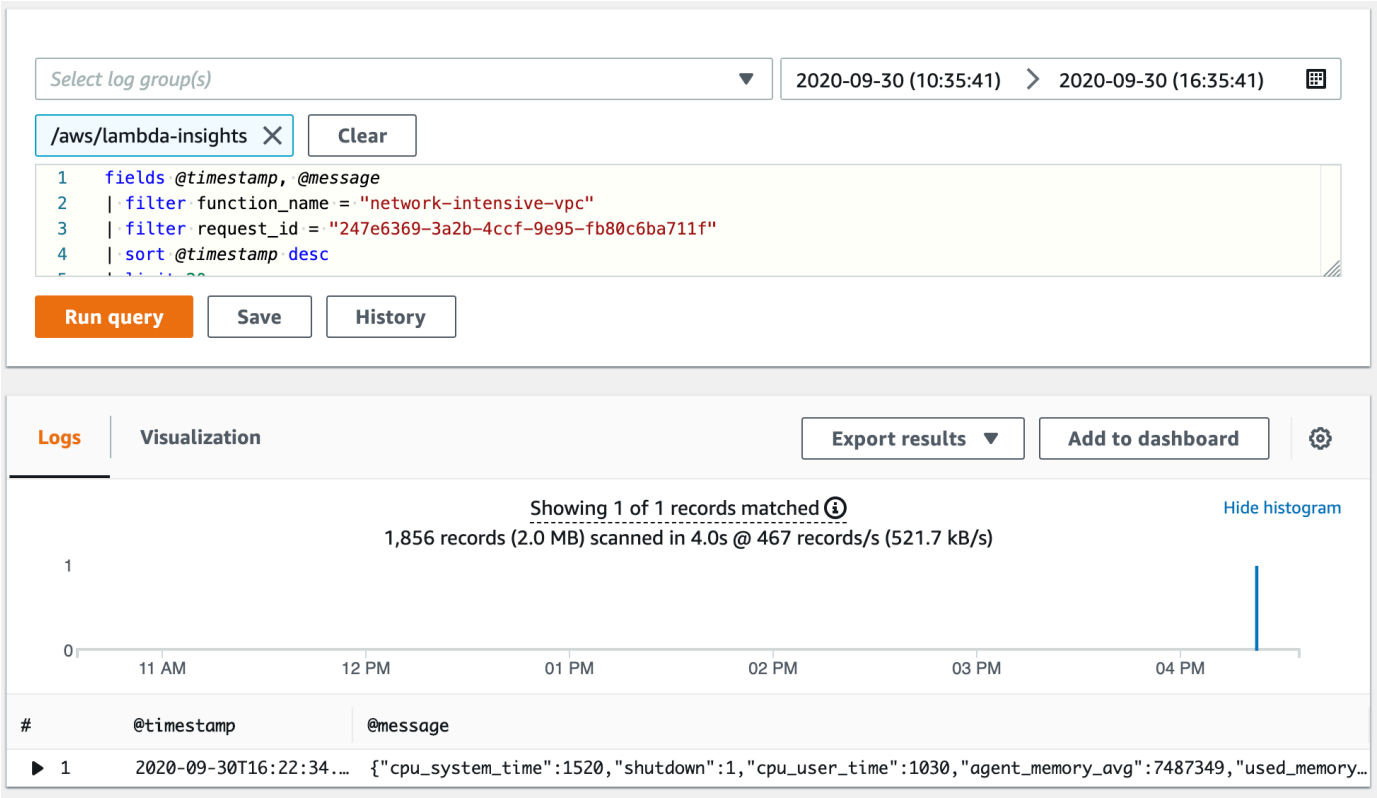
7. Untuk menampilkan 1000 invokasi terbaru, pilih tab Invokasi.
8. Pilih Stempel waktu atau Pesan untuk permintaan invokasi yang ingin Anda pecahkan.



9. Pilih daftar pilihan menurun Lihat log, lalu pilih Lihat log kinerja.

   Kueri yang dibuat secara otomatis untuk Fungsi Acara Anda terbuka di dasbor Wawasan Log.

10. Pilih Jalankan kueri untuk menghasilkan pesan Log untuk permintaan invokasi.



Apa selanjutnya?

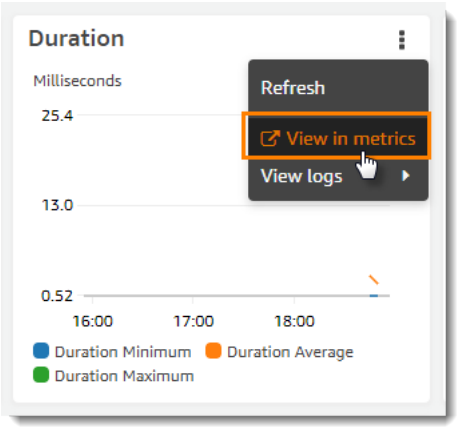
- Pelajari cara membuat dasbor CloudWatch Log di Membuat Dasbor di Panduan CloudWatch Pengguna Amazon.
- Pelajari cara menambahkan kueri ke dasbor CloudWatch Log di Tambahkan Kueri ke Dasbor atau Ekspor Hasil Kueri di Panduan CloudWatch Pengguna Amazon.

Memantau aplikasi Lambda

Bagian Aplikasi pada konsol Lambda menyertakan tab Pemantauan tempat Anda dapat meninjau CloudWatch dasbor Amazon dengan metrik agregat untuk sumber daya dalam aplikasi Anda.

Untuk memantau aplikasi Lambda

1. Buka konsol halaman Aplikasi konsol Lambda.
2. Pilih Pemantauan.
3. Untuk melihat detail selengkapnya tentang metrik dalam grafik apa pun, pilih Lihat dalam metrik dari menu tarik-turun.



   Grafik tersebut kemudian akan muncul di sebuah tab baru yang menampilkan metrik-metrik yang relevan yang tercantum di bawah grafik tersebut. Anda dapat menyesuaikan tampilan grafik ini, juga mengubah metrik-metrik dan sumber daya yang ditampilkan, statistik, periode, dan faktor lainnya yang ditampilkan untuk mendapatkan pemahaman lebih baik tentang situasi saat ini.

Secara default, konsol Lambda menunjukkan dasbor dasar. Anda dapat menyesuaikan halaman ini dengan menambahkan satu atau beberapa CloudWatch dasbor Amazon ke template aplikasi Anda dengan tipe sumber daya AWS::CloudWatch: :Dashboard. Ketika templat Anda mencakup satu dasbor atau lebih, halaman tersebut menampilkan dasbor Anda alih-alih dasbor default. Anda dapat beralih antara dasbor dengan menu tarik turun di kanan atas halaman. Contoh berikut membuat dashboard dengan widget tunggal yang grafik jumlah pemanggilan dari Fungsi Acara bernama. `my-function`

Example Templat dasbor Fungsi Acara

```
Resources:
```

```
  MyDashboard:
    Type: AWS::CloudWatch::Dashboard
    Properties:
      DashboardName: my-dashboard
      DashboardBody: |
        {
            "widgets": [
                {
                    "type": "metric",
                    "width": 12,
                    "height": 6,
                    "properties": {
                        "metrics": [
                            [
                                "AWS/Lambda",
                                "Invocations",
                                "FunctionName",
                                "my-function",
                                {
                                    "stat": "Sum",
                                    "label": "MyFunction"
                                }
                            ],
                            [
                                {
                                    "expression": "SUM(METRICS())",
                                    "label": "Total Invocations"
                                }
                            ]
                        ],
                        "region": "us-east-1",
                        "title": "Invocations",
                        "view": "timeSeries",
                        "stacked": false
                    }
                }
            ]
        }
```

Untuk informasi selengkapnya tentang pembuatan CloudWatch dasbor dan widget, lihat Struktur dan sintaks isi dasbor di Referensi Amazon API. CloudWatch

Pantau kinerja aplikasi dengan Sinyal CloudWatch Aplikasi Amazon

Amazon CloudWatch Application Signals adalah solusi pemantauan kinerja aplikasi (APM) yang memungkinkan pengembang dan operator memantau kesehatan dan kinerja aplikasi tanpa server mereka yang dibuat menggunakan Lambda. Anda dapat mengaktifkan Sinyal Aplikasi dalam sekali klik dari konsol Lambda, dan Anda tidak perlu menambahkan kode instrumentasi atau dependensi eksternal apa pun ke Fungsi Peristiwa Lambda Anda. Setelah mengaktifkan Sinyal Aplikasi, Anda dapat melihat semua metrik dan jejak yang dikumpulkan di CloudWatch konsol. Halaman ini menjelaskan cara mengaktifkan dan melihat data telemetri Sinyal Aplikasi untuk aplikasi Anda.

Topik

- Bagaimana Sinyal Aplikasi terintegrasi dengan Lambda
- Harga
- Waktu aktif yang didukung
- Mengaktifkan Sinyal Aplikasi di konsol Lambda
- Menggunakan dasbor Sinyal Aplikasi

Bagaimana Sinyal Aplikasi terintegrasi dengan Lambda

Sinyal Aplikasi secara otomatis menginstrumentasikan Fungsi Peristiwa Lambda Anda menggunakan pustaka AWS Distro for OpenTelemetry (ADOT) yang disempurnakan, yang disediakan melalui lapisan Lambda. Sinyal Aplikasi membaca data yang dikumpulkan oleh lapisan dan menghasilkan dasbor dengan metrik kinerja utama untuk aplikasi Anda.

Anda dapat melampirkan layer ini dalam satu klik dengan mengaktifkan Sinyal Aplikasi di konsol Lambda. Saat Anda mengaktifkan Sinyal Aplikasi dari konsol, Lambda melakukan hal berikut atas nama Anda:

- Memperbarui peran eksekusi Event Function Anda untuk menyertakan`CloudWatchLambdaApplicationSignalsExecutionRolePolicy`. Kebijakan ini menyediakan akses tulis ke AWS X-Ray dan grup CloudWatch log yang digunakan untuk Sinyal Aplikasi.
- Menambahkan lapisan ke Fungsi Acara Anda yang secara otomatis instrumen Fungsi Acara untuk menangkap data telemetri seperti permintaan, ketersediaan, latensi, kesalahan, dan kesalahan. Untuk memastikan bahwa Sinyal Aplikasi berfungsi dengan baik, hapus kode instrumentasi X-Ray

SDK yang ada dari Fungsi Acara Anda. Kode instrumentasi X-Ray SDK khusus dapat mengganggu instrumentasi yang disediakan lapisan.

- Menambahkan variabel `AWS_LAMBDA_EXEC_WRAPPER` lingkungan ke Fungsi Acara Anda, dan menetapkan nilainya`/opt/otel-instrument`. Variabel lingkungan ini memodifikasi perilaku startup Event Function Anda untuk menggunakan lapisan Application Signals, dan diperlukan untuk instrumentasi yang tepat. Jika variabel lingkungan ini sudah ada, pastikan bahwa itu disetel ke nilai yang diperlukan.

Harga

Menggunakan Sinyal Aplikasi untuk Fungsi Acara Lambda Anda menimbulkan biaya. Untuk informasi harga, lihat CloudWatch harga Amazon.

Waktu aktif yang didukung

Integrasi Sinyal Aplikasi dengan Lambda berfungsi dengan runtime berikut:

- .NET 8
- Java 11
- Jawa 17
- Jawa 21
- Python 3.10
- Python 3.11
- Python 3.12
- Python 3.13
- Node.js 18.x
- Node.js 20.x
- Node.js 22.x

Mengaktifkan Sinyal Aplikasi di konsol Lambda

Anda dapat mengaktifkan Sinyal Aplikasi pada Fungsi Peristiwa Lambda yang ada menggunakan runtime yang didukung. Langkah-langkah berikut menjelaskan cara mengaktifkan Sinyal Aplikasi dalam satu klik di konsol Lambda.

Untuk mengaktifkan Sinyal Aplikasi di konsol Lambda

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih Fungsi Acara Anda.
3. Pilih tab Konfigurasi.
4. Di menu sebelah kiri, pilih Alat pemantauan dan operasi.
5. Pada panel Alat pemantauan tambahan, pilih Edit.
6. Di bawah Sinyal CloudWatch Aplikasi dan AWS X-Ray, dan di bawah Sinyal Aplikasi, pilih Aktifkan.
7. Pilih Simpan.

Jika ini adalah pertama kalinya Anda mengaktifkan Sinyal Aplikasi untuk Fungsi Acara Anda, Anda juga harus melakukan penyiapan penemuan layanan satu kali untuk Sinyal Aplikasi di CloudWatch konsol. Setelah Anda menyelesaikan penyiapan penemuan layanan satu kali ini, Sinyal Aplikasi secara otomatis menemukan Fungsi Peristiwa Lambda tambahan yang Anda aktifkan untuk Sinyal Aplikasi, di semua Wilayah.

> **Note**
>
> Setelah Anda memanggil Fungsi Acara yang diperbarui, diperlukan waktu hingga 10 menit agar data layanan mulai muncul di dasbor Sinyal Aplikasi di CloudWatch konsol.

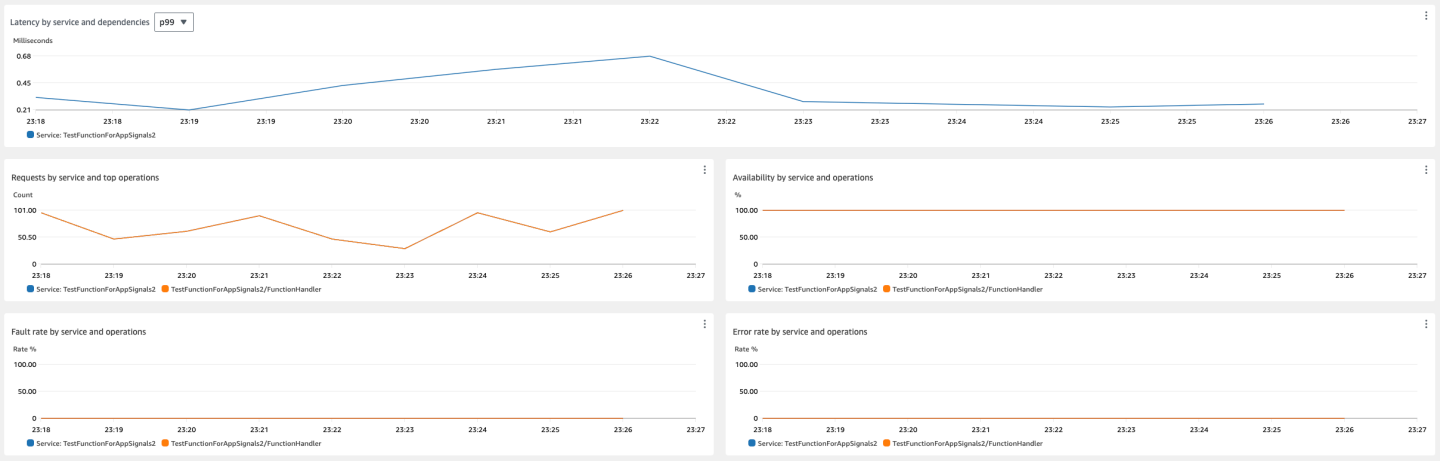
Menggunakan dasbor Sinyal Aplikasi

Setelah mengaktifkan Sinyal Aplikasi untuk Fungsi Acara, Anda dapat memvisualisasikan metrik aplikasi di CloudWatch konsol. Anda dapat dengan cepat melihat dasbor Sinyal Aplikasi terkait dari konsol Lambda dengan langkah-langkah berikut:

Untuk melihat dasbor Sinyal Aplikasi untuk Fungsi Acara Anda

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih Fungsi Acara Anda.
3. Pilih tab Pantau.
4. Pilih tombol Lihat Sinyal Aplikasi. Ini akan membawa Anda langsung ke ikhtisar Sinyal Aplikasi untuk layanan Anda di CloudWatch konsol.

Misalnya, tangkapan layar berikut menunjukkan metrik untuk latensi, jumlah permintaan, ketersediaan, tingkat kesalahan, dan tingkat kesalahan untuk Fungsi Peristiwa di seluruh jendela waktu 10 menit.



Untuk memaksimalkan integrasi Anda dengan Sinyal Aplikasi, Anda dapat membuat tujuan tingkat layanan (SLOS) untuk aplikasi Anda. Misalnya, Anda dapat membuat SL latensi untuk memastikan aplikasi Anda merespons permintaan pengguna dengan cepat, dan SL ketersediaan untuk melacak waktu aktif. SLOS dapat membantu Anda mendeteksi penurunan kinerja atau pemadaman sebelum berdampak pada pengguna Anda. Untuk informasi selengkapnya, lihat Tujuan tingkat layanan (SLOS) di Panduan CloudWatch Pengguna Amazon.

Debug dari jarak jauh Fungsi Acara Lambda dengan Kode Visual Studio

Dengan fitur debugging jarak jauh di AWS Toolkit for Visual Studio Code, Anda dapat men-debug Fungsi Peristiwa Lambda yang berjalan langsung di cloud. AWS Ini berguna saat menyelidiki masalah yang sulit direplikasi secara lokal atau didiagnosis hanya dengan log.

Dengan debugging jarak jauh, Anda dapat:

- Tetapkan breakpoint dalam kode Fungsi Peristiwa Lambda Anda.
- Melangkah melalui eksekusi kode secara real-time.
- Periksa variabel dan status selama runtime.
- Debug Lambda Event Functions yang digunakan AWS, termasuk yang ada di VPC atau dengan izin IAM tertentu.

Waktu aktif yang didukung

Debugging jarak jauh didukung untuk runtime berikut:

- Python (AL2023)
- Java
- JavaScript/Node.js (AL2023)

> **Note**
>
> Debugging jarak jauh didukung untuk arsitektur x86_64 dan arm64.

Keamanan dan debugging jarak jauh

Debugging jarak jauh beroperasi dalam batas keamanan Lambda yang ada. Pengguna dapat melampirkan lapisan ke Fungsi Acara menggunakan `UpdateFunctionConfiguration` izin, yang sudah memiliki kemampuan untuk mengakses variabel dan konfigurasi lingkungan Fungsi Acara. Debugging jarak jauh tidak melampaui izin yang ada ini. Sebaliknya, ia menambahkan kontrol keamanan ekstra melalui tunneling aman dan manajemen sesi otomatis. Selain itu, debugging jarak jauh sepenuhnya merupakan fitur yang dikendalikan pelanggan yang memerlukan izin dan tindakan eksplisit:

- Pembuatan Terowongan Aman IoT: AWS Toolkit harus membuat terowongan aman IoT, yang hanya terjadi dengan izin eksplisit pengguna menggunakan. `iot:OpenTunnel`
- Debug Layer Attachment dan Token Management: Proses debugging menjaga keamanan melalui kontrol ini:
  - Lapisan debugging harus dilampirkan ke Fungsi Peristiwa Lambda dan proses ini memerlukan izin `lambda:UpdateFunctionConfiguration` berikut: dan. `lambda:GetLayerVersion`
  - Perbarui token keamanan (dihasilkan melalui`iot:OpenTunnel`) dalam variabel lingkungan Fungsi Acara sebelum setiap sesi debug. Langkah ini juga membutuhkan`lambda:UpdateFunctionConfiguration`.
  - Untuk keamanan, token ini secara otomatis diputar dan lapisan debug secara otomatis dihapus pada akhir setiap sesi debug dan tidak dapat digunakan kembali.

> **Note**
>
> Debugging jarak jauh didukung untuk arsitektur x86_64 dan arm64.

Prasyarat

Sebelum Anda memulai debugging jarak jauh, pastikan Anda memiliki yang berikut:

1. Fungsi Peristiwa Lambda diterapkan ke akun Anda. AWS
2. AWS Toolkit for Visual Studio Code. Lihat Menyiapkan AWS Toolkit for Visual Studio Code petunjuk penginstalan.
3. Versi AWS Toolkit yang telah Anda instal adalah 3.69.0 atau yang lebih baru.
4. AWS kredensional dikonfigurasi di. AWS Toolkit for Visual Studio Code Untuk informasi selengkapnya, lihat Kontrol autentikasi dan akses.

Debug dari jarak jauh Fungsi Acara Lambda

Ikuti langkah-langkah ini untuk memulai sesi debugging jarak jauh:

1. Buka AWS Explorer di VS Code dengan memilih AWS ikon di sidebar kiri.
2. Perluas bagian Lambda untuk melihat Fungsi Acara Anda.
3. Buka menu konteks untuk Fungsi Acara yang ingin Anda debug.

4. Dari menu konteks, pilih Memanggil dari jarak jauh.
5. Di jendela pemanggilan yang terbuka, centang kotak untuk Aktifkan debugging.
6. Klik Memanggil untuk memulai sesi debugging jarak jauh.

> **Note**
>
> Lambda Event Functions memiliki batas gabungan 250MB untuk kode Fungsi Acara dan semua lapisan terlampir. Lapisan debugging jarak jauh menambahkan sekitar 40MB ke ukuran Event Function Anda.

Sesi debugging jarak jauh berakhir saat Anda:

- Pilih Hapus Pengaturan Debug dari layar konfigurasi panggilan jarak jauh.
- Pilih ikon putuskan sambungan di kontrol debugging VS Code.
- Pilih file handler di editor VS Code.

> **Note**
>
> Lapisan debug secara otomatis dihapus setelah 60 detik tidak aktif setelah pemanggilan terakhir Anda.

Nonaktifkan debugging jarak jauh

Ada tiga cara untuk menonaktifkan fitur ini:

- Tolak Pembaruan Fungsi: Setel `lambda:UpdateFunctionConfiguration` kedeny.
- Batasi Izin IoT: Tolak izin IoT-related
- Blokir Lapisan Debug: Tolak `lambda:GetLayerVersion` untuk ARN berikut:
  - `arn:aws:lambda:*:*:layer:LDKLayerX86:*`
  - `arn:aws:lambda:*:*:layer:LDKLayerArm64:*`

> **Note**
>
> Menonaktifkan fitur ini mencegah lapisan debugging ditambahkan selama pembaruan konfigurasi Fungsi Acara.

Informasi tambahan

Untuk informasi lebih lanjut tentang penggunaan Lambda di VS Code, lihat Mengembangkan fungsi Lambda secara lokal dengan VS Code.

Untuk petunjuk mendetail tentang pemecahan masalah, kasus penggunaan lanjutan, dan ketersediaan wilayah, lihat Fungsi Lambda debugging jarak jauh di Panduan Pengguna. AWS Toolkit for Visual Studio Code

Memanggil Lambda dengan acara dari yang lain AWS layanan

Beberapa Layanan AWS dapat langsung memanggil Lambda Event Functions menggunakan pemicu. Layanan ini mendorong peristiwa ke Lambda, dan fungsi segera dipanggil saat peristiwa yang ditentukan terjadi. Pemicu cocok untuk peristiwa diskrit dan pemrosesan waktu nyata. Saat Anda membuat pemicu menggunakan konsol Lambda, konsol berinteraksi dengan AWS layanan terkait untuk mengonfigurasi pemberitahuan peristiwa pada layanan tersebut. Pemicunya sebenarnya disimpan dan dikelola oleh layanan yang menghasilkan peristiwa, bukan oleh Lambda.

Peristiwa adalah data yang terstruktur dalam format JSON. Struktur JSON bervariasi tergantung pada layanan yang menghasilkannya dan jenis acara, tetapi semuanya berisi data yang dibutuhkan fungsi untuk memproses acara.

Selain pemanggilan berbasis peristiwa, Anda dapat menggunakan Lambda dengan Amazon EC2, Apache Kafka yang dikelola sendiri, dan Kubernetes.

Fungsi Peristiwa dapat memiliki beberapa pemicu. Setiap pemicu bertindak sebagai klien yang menjalankan fungsi Anda secara independen, dan setiap peristiwa yang diteruskan Lambda ke fungsi Anda hanya memiliki data dari satu pemicu. Lambda mengubah dokumen acara menjadi objek dan meneruskannya ke penangan fungsi Anda.

Bergantung pada layanannya, pemanggilan berbasis peristiwa dapat sinkron atau asinkron.

- Untuk pemanggilan sinkron, layanan yang menghasilkan acara menunggu respons dari fungsi Anda. Layanan itu mendefinisikan data yang perlu dikembalikan fungsi dalam respons. Layanan mengontrol strategi kesalahan, seperti apakah akan mencoba lagi kesalahan.
- Untuk invokasi asinkron, Lambda mengantrekan peristiwa sebelum memberikannya ke fungsi Anda. Ketika Lambda mengantri acara, ia segera mengirimkan respons sukses ke layanan yang menghasilkan acara tersebut. Setelah fungsi memproses peristiwa, Lambda tidak mengembalikan respons ke layanan penghasil peristiwa.

Membuat pemicu

Cara termudah untuk membuat pemicu adalah dengan menggunakan konsol Lambda. Saat Anda membuat pemicu menggunakan konsol, Lambda secara otomatis menambahkan izin yang diperlukan ke kebijakan berbasis sumber daya Fungsi Acara.

Untuk membuat pemicu menggunakan konsol Lambda

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih fungsi yang ingin Anda buat pemicu.
3. Di panel Ikhtisar fungsi, pilih Tambah pemicu.
4. Pilih AWS layanan yang ingin Anda panggil fungsi Anda.
5. Isi opsi di panel konfigurasi Pemicu dan pilih Tambah. Bergantung pada pilihan Layanan AWS Anda untuk menjalankan fungsi Anda, opsi konfigurasi pemicu berbeda.

Layanan yang dapat memanggil Fungsi Acara Lambda

Tabel berikut mencantumkan layanan yang dapat memanggil Fungsi Peristiwa Lambda.

Layanan	Metode doa
Amazon Managed Streaming for Apache Kafka	Pemetaan sumber acara
Self-managed Apache Kafka	Pemetaan sumber acara
Amazon API Gateway	Event-driven; doa sinkron

Layanan	Metode doa
AWS CloudFormation	Event-driven; doa asinkron
CloudWatch Log Amazon	Event-driven; doa asinkron
AWS CodeCommit	Event-driven; doa asinkron
AWS CodePipeline	Event-driven; doa asinkron
Amazon Cognito	Event-driven; doa sinkron
AWS Config	Event-driven; doa asinkron
Connect Pelanggan	Event-driven; doa sinkron
Amazon DocumentDB	Pemetaan sumber acara
Amazon DynamoDB	Pemetaan sumber acara
Elastic Load Balancing (Application Load Balancer)	Event-driven; doa sinkron
EventBridge	Event-driven; pemanggilan asinkron (bus dan aturan acara), pemanggilan sinkron atau asinkron (pipa)
EventBridge Penjadwal Amazon	Event-driven (berbasis waktu); pemanggilan asinkron
AWS IoT	Event-driven; doa asinkron
Amazon Kinesis	Pemetaan sumber acara
Amazon Data Firehose	Event-driven; doa sinkron
Amazon Lex	Event-driven; doa sinkron
Amazon MQ	Pemetaan sumber acara
Layanan Email Amazon Sederhana	Event-driven; doa asinkron

Layanan	Metode doa
Layanan Pemberitahuan Sederhana Amazon	Event-driven; doa asinkron
Amazon Simple Queue Service	Pemetaan sumber acara
Amazon Simple Storage Service (Amazon S3)	Event-driven; doa asinkron
Batch Layanan Penyimpanan Sederhana Amazon	Event-driven; doa sinkron
Secrets Manager	Rotasi rahasia
AWS Step Functions	Event-driven; doa sinkron atau asinkron
Kisi VPC Amazon	Event-driven; doa sinkron

Menggunakan Lambda dengan Apache Kafka

Lambda mendukung Apache Kafka sebagai sumber peristiwa. Apache Kafka adalah platform streaming acara sumber terbuka yang dirancang untuk menangani throughput tinggi, saluran data real-time, dan aplikasi streaming. Ada dua cara utama untuk menggunakan Lambda dengan Apache Kafka:

- the section called “MSK” — Amazon Managed Streaming untuk Apache Kafka (Amazon MSK) adalah layanan yang dikelola sepenuhnya oleh AWS. Amazon MSK membantu mengotomatiskan pengelolaan infrastruktur Kafka Anda, termasuk penyediaan, tambalan, dan penskalaan.
- the section called “Self-managed Apache Kafka” — Dalam AWS terminologi, cluster yang dikelola sendiri mencakup non-AWS menghosting cluster Kafka. Misalnya, Anda masih dapat menggunakan Lambda dengan cluster Kafka yang dihosting dengan non-AWS Penyedia cloud seperti Awan Bersambungan atau Red Panda.

Saat memutuskan antara Amazon MSK dan Apache Kafka yang dikelola sendiri, pertimbangkan kebutuhan operasional dan persyaratan kontrol Anda. Amazon MSK adalah pilihan yang lebih baik jika Anda mau AWS untuk membantu Anda mengelola pengaturan Kafka yang dapat diskalakan dan siap produksi dengan biaya operasional minimal. Ini menyederhanakan keamanan, pemantauan, dan ketersediaan tinggi, membantu Anda fokus pada pengembangan aplikasi daripada manajemen infrastruktur. Di sisi lain, Apache Kafka yang dikelola sendiri lebih cocok untuk kasus penggunaan yang berjalan pada non-#-AWS lingkungan yang dihosting, termasuk cluster lokal.

Topik

- Menggunakan Lambda dengan Amazon MSK
- Menggunakan Lambda dengan Apache Kafka yang dikelola sendiri
- Mode penskalaan poller acara Apache Kafka di Lambda
- Polling Apache Kafka dan posisi awal streaming di Lambda
- Polling topik Kafka dalam mode Antrian
- ID grup konsumen yang dapat disesuaikan di Lambda
- Memfilter acara dari Amazon MSK dan sumber acara Apache Kafka yang dikelola sendiri
- Menggunakan pendaftar skema dengan sumber acara Kafka di Lambda
- Pemrosesan latensi rendah untuk sumber acara Kafka
- Mengonfigurasi kontrol penanganan kesalahan untuk sumber acara Kafka

- Menangkap batch yang dibuang untuk Amazon MSK dan sumber acara Apache Kafka yang dikelola sendiri
- Menggunakan topik Kafka sebagai tujuan kegagalan
- Pencatatan pemetaan sumber acara Kafka
- Memecahkan masalah kesalahan pemetaan sumber acara Kafka

Menggunakan Lambda dengan Amazon MSK

Amazon Managed Streaming for Apache Kafka (Amazon MSK) adalah layanan yang dikelola sepenuhnya yang dapat Anda gunakan untuk membangun dan menjalankan aplikasi yang menggunakan Apache Kafka untuk memproses data streaming. Amazon MSK menyederhanakan pengaturan, penskalaan, dan pengelolaan cluster Kafka. Amazon MSK juga memudahkan untuk mengkonfigurasi aplikasi Anda untuk beberapa Availability Zone dan untuk keamanan dengan AWS Identity and Access Management (IAM).

Bab ini menjelaskan cara menggunakan kluster MSK Amazon sebagai sumber acara untuk Fungsi Acara Lambda Anda. Proses umum untuk mengintegrasikan Amazon MSK dengan Lambda melibatkan langkah-langkah berikut:

1. Pengaturan kluster dan jaringan — Pertama, siapkan kluster MSK Amazon Anda. Ini termasuk konfigurasi jaringan yang benar untuk memungkinkan Lambda mengakses cluster Anda.
2. Pengaturan pemetaan sumber peristiwa — Kemudian, buat sumber daya pemetaan sumber peristiwa yang dibutuhkan Lambda untuk menghubungkan kluster MSK Amazon Anda dengan aman ke fungsi Anda.
3. Fungsi Acara dan pengaturan izin — Terakhir, pastikan bahwa fungsi Anda diatur dengan benar, dan memiliki izin yang diperlukan dalam peran pelaksanaannya.

> **Note**
>
> Anda sekarang dapat membuat dan mengelola pemetaan sumber acara MSK Amazon Anda langsung dari konsol Lambda atau Amazon MSK. Kedua konsol menawarkan opsi untuk secara otomatis menangani pengaturan izin peran eksekusi Lambda yang diperlukan untuk proses konfigurasi yang lebih efisien.

Untuk contoh tentang cara menyiapkan integrasi Lambda dengan kluster MSK Amazon, lihat, the section called “Tutorial” Menggunakan Amazon MSK sebagai sumber acara di AWS Blog Komputasi, dan Integrasi Lambda MSK Amazon AWS Lambda di Labs MSK Amazon.

Topik

- Contoh peristiwa
- Mengonfigurasi kluster MSK Amazon dan jaringan VPC Amazon untuk Lambda
- Mengonfigurasi izin Lambda untuk pemetaan sumber acara MSK Amazon

- Mengonfigurasi sumber acara MSK Amazon untuk Lambda
- Tutorial: Menggunakan pemetaan sumber acara MSK Amazon untuk menjalankan Fungsi Acara Lambda

Contoh peristiwa

Lambda mengirimkan kumpulan pesan dalam parameter peristiwa saat memanggil Fungsi Acara Anda. Muatan peristiwa berisi array pesan. Setiap item array berisi detail dari topik Amazon MSK dan pengidentifikasi partisi, bersama-sama dengan stempel waktu dan pesan berkode base64.


```
{
   "eventSource":"aws:kafka",
   "eventSourceArn":"arn:aws:kafka:us-east-1:123456789012:cluster/
vpc-2priv-2pub/751d2973-a626-431c-9d4e-d7975eb44dd7-2",
   "bootstrapServers":"b-2.demo-cluster-1.a1bcde.c1.kafka.us-
east-1.amazonaws.com:9092,b-1.demo-cluster-1.a1bcde.c1.kafka.us-
east-1.amazonaws.com:9092",
   "records":{
      "mytopic-0":[
         {
            "topic":"mytopic",
            "partition":0,
            "offset":15,
            "timestamp":1545084650987,
            "timestampType":"CREATE_TIME",
            "key":"abcDEFghiJKLmnoPQRstuVWXyz1234==",
            "value":"SGVsbG8sIHRoaXMgaXMgYSB0ZXN0Lg==",
            "headers":[
               {
                  "headerKey":[
                     104,
                     101,
                     97,
                     100,
                     101,
                     114,
                     86,
                     97,
                     108,
                     117,
                     101
                  ]
```

```
                }
            ]
        }
    ]
  }
}
```

Mengonfigurasi kluster MSK Amazon dan jaringan VPC Amazon untuk Lambda

Untuk menghubungkan Fungsi AWS Lambda Acara ke kluster MSK Amazon, Anda perlu mengonfigurasi klaster dengan benar dan Amazon Virtual Private Cloud (VPC) tempat ia berada. Halaman ini menjelaskan cara mengkonfigurasi cluster dan VPC Anda. Jika cluster dan VPC Anda sudah dikonfigurasi dengan benar, lihat the section called “Konfigurasikan sumber acara” untuk mengonfigurasi pemetaan sumber peristiwa.

Topik

- Ikhtisar persyaratan konfigurasi jaringan untuk integrasi Lambda dan MSK
- Mengkonfigurasi gateway NAT untuk sumber acara MSK
- Melakukan konfigurasi AWS PrivateLink titik akhir untuk sumber acara MSK

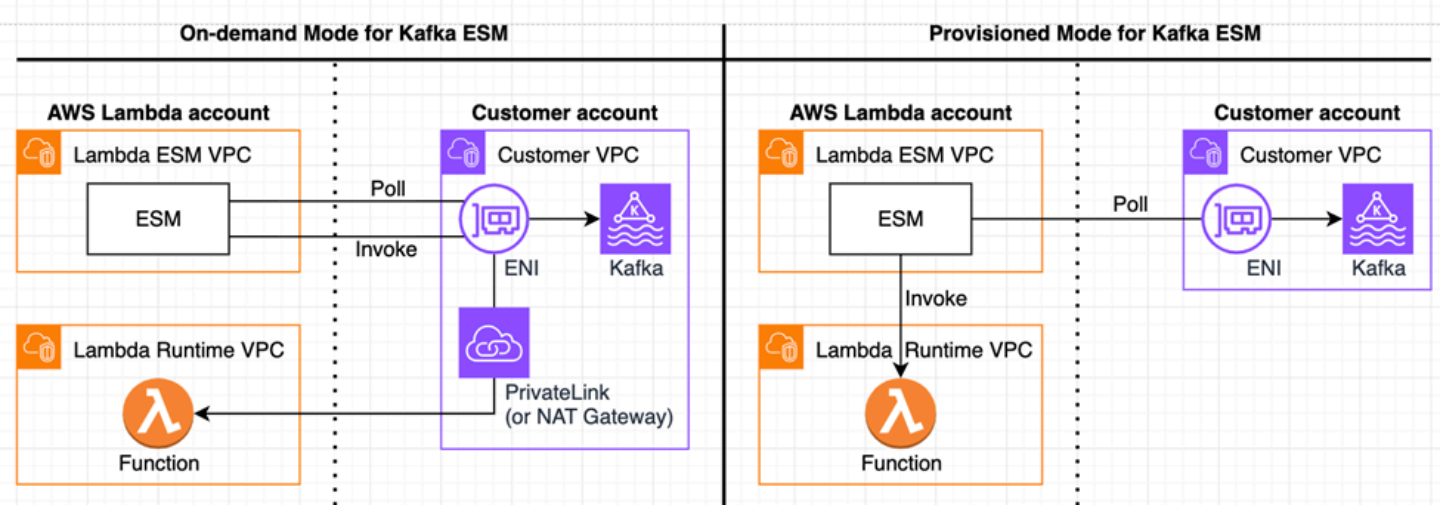
Ikhtisar persyaratan konfigurasi jaringan untuk integrasi Lambda dan MSK

Konfigurasi jaringan yang diperlukan untuk integrasi Lambda dan MSK bergantung pada arsitektur jaringan aplikasi Anda. Ada tiga sumber daya utama yang terlibat dalam integrasi ini: cluster MSK Amazon, Fungsi Acara Lambda, dan pemetaan sumber peristiwa Lambda. Masing-masing sumber daya ini berada di VPC yang berbeda:

- Cluster MSK Amazon Anda biasanya berada di subnet pribadi VPC yang Anda kelola.
- Fungsi Acara Lambda Anda berada di VPC terkelola yang dimiliki oleh AWS Lambda.
- Pemetaan sumber peristiwa Lambda Anda berada di AWS VPC terkelola lain yang dimiliki oleh Lambda, terpisah dari VPC yang berisi fungsi Anda.

Pemetaan sumber peristiwa adalah sumber daya perantara antara kluster MSK dan Fungsi Peristiwa Lambda. Pemetaan sumber acara memiliki dua pekerjaan utama. Pertama, ia polling cluster MSK Anda untuk pesan baru. Kemudian, ia memanggil Fungsi Acara Lambda Anda dengan pesan-pesan itu. Karena ketiga sumber daya ini berada di VPC yang berbeda, operasi polling dan pemanggilan memerlukan panggilan jaringan lintas-VPC.

Persyaratan konfigurasi jaringan untuk pemetaan sumber peristiwa Anda bergantung pada apakah menggunakan mode yang disediakan atau mode sesuai permintaan, seperti yang ditunjukkan pada diagram berikut:



Cara pemetaan sumber peristiwa Lambda melakukan polling pada cluster MSK Anda untuk pesan baru adalah sama di kedua mode. Untuk membuat koneksi antara pemetaan sumber peristiwa dan kluster MSK Anda, Lambda membuat ENI hyperplane (atau menggunakan kembali yang sudah ada, jika tersedia) di subnet pribadi Anda untuk membuat koneksi yang aman. Seperti yang diilustrasikan dalam diagram, ENI hyperplane ini menggunakan konfigurasi subnet dan grup keamanan klaster MSK Anda, bukan Fungsi Peristiwa Lambda Anda.

Setelah polling pesan dari cluster, cara Lambda memanggil fungsi Anda berbeda di setiap mode:

- Dalam mode yang disediakan, Lambda secara otomatis menangani koneksi antara VPC pemetaan sumber peristiwa dan fungsi VPC. Jadi, Anda tidak memerlukan komponen jaringan tambahan untuk berhasil menjalankan fungsi Anda.
- Dalam mode sesuai permintaan, pemetaan sumber peristiwa Lambda Anda memanggil fungsi Anda dengan menggunakan jalur melalui VPC yang dikelola pelanggan. Karena itu, Anda perlu mengonfigurasi gateway NAT di subnet publik VPC Anda, atau AWS PrivateLinktitik akhir di subnet pribadi VPC yang menyediakan akses ke Lambda, (STS), dan secara opsional,.AWS Security Token ServiceAWS Secrets Manager Mengkonfigurasi salah satu dari opsi ini dengan benar memungkinkan koneksi antara VPC Anda dan VPC Lambda-managed runtime, yang diperlukan untuk menjalankan fungsi Anda.

Gateway NAT memungkinkan sumber daya di subnet pribadi Anda untuk mengakses internet publik. Menggunakan konfigurasi ini berarti lalu lintas Anda melintasi internet sebelum menjalankan Fungsi

Peristiwa Lambda. AWS PrivateLink endpoint memungkinkan subnet pribadi terhubung dengan aman ke AWS layanan atau sumber VPC pribadi lainnya tanpa melintasi internet publik. Lihat the section called “Mengkonfigurasi gateway NAT untuk sumber acara MSK” atau the section called “Melakukan konfigurasi AWS PrivateLink titik akhir untuk sumber acara MSK” untuk detail tentang cara mengonfigurasi sumber daya ini.

Sejauh ini, kami berasumsi bahwa klaster MSK Anda berada di subnet pribadi dalam VPC Anda, yang merupakan kasus yang lebih umum. Namun, bahkan jika klaster MSK Anda berada di subnet publik dalam VPC Anda, Anda harus mengonfigurasi AWS PrivateLink titik akhir untuk mengaktifkan koneksi yang aman. Tabel berikut merangkum persyaratan konfigurasi jaringan berdasarkan cara Anda mengonfigurasi kluster MSK dan pemetaan sumber peristiwa Lambda:

Lokasi klaster MSK (dalam VPC yang dikelola pelanggan)	Mode penskalaan pemetaan sumber acara Lambda	Konfigurasi jaringan yang diperlukan
Subnet privat	On-demand modus	Gateway NAT (di subnet publik VPC Anda), atau AWS PrivateLink titik akhir (di subnet pribadi VPC Anda) untuk mengaktifkan akses ke Lambda,, dan secara opsional, Secrets Manager. AWS STS
Subnet publik	On-demand modus	AWS PrivateLink endpoint (di subnet publik VPC Anda) untuk mengaktifkan akses ke Lambda,, dan secara opsional AWS STS, Secrets Manager.
Subnet privat	Mode yang disediakan	Tidak ada
Subnet publik	Mode yang disediakan	Tidak ada

Selain itu, grup keamanan yang terkait dengan kluster MSK Anda harus mengizinkan lalu lintas melalui port yang benar. Pastikan Anda memiliki aturan grup keamanan berikut yang dikonfigurasi:

- Aturan masuk - Izinkan semua lalu lintas di port broker default. Port yang digunakan MSK tergantung pada jenis otentikasi pada cluster: `9098` untuk otentikasi IAM, untuk, dan `9096` untuk

SASL/SCRAM TLS. 9094 Atau, Anda dapat menggunakan aturan grup keamanan referensi mandiri untuk mengizinkan akses dari instans dalam grup keamanan yang sama.

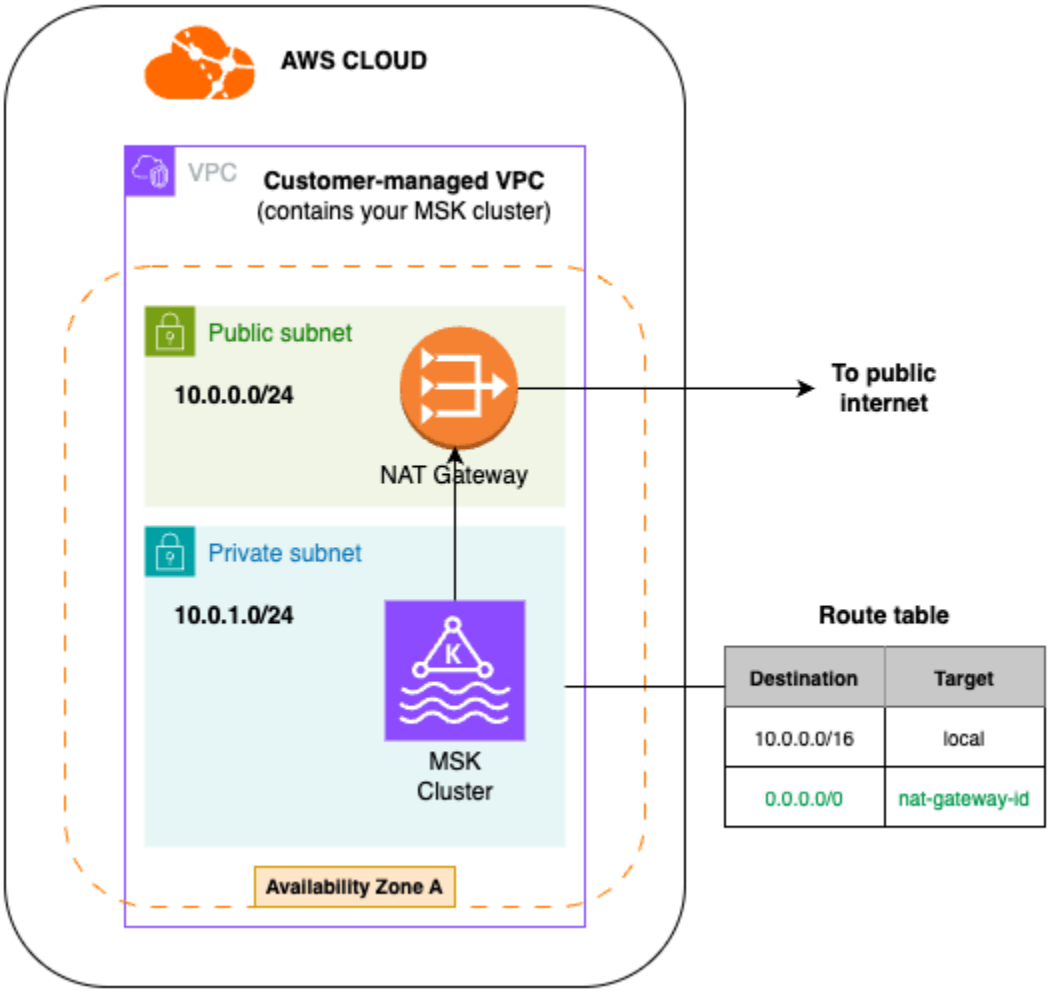
- Aturan keluar — Izinkan semua lalu lintas di port 443 untuk tujuan eksternal jika fungsi Anda perlu berkomunikasi dengan AWS layanan lain. Atau, Anda dapat menggunakan aturan grup keamanan referensi diri untuk membatasi akses ke broker jika Anda tidak perlu berkomunikasi dengan layanan lain AWS .
- Aturan masuk titik akhir Amazon VPC — Jika Anda menggunakan titik akhir VPC Amazon, grup keamanan yang terkait dengan titik akhir harus mengizinkan lalu lintas masuk pada port dari grup keamanan klaster. 443

Mengkonfigurasi gateway NAT untuk sumber acara MSK

Anda dapat mengonfigurasi gateway NAT untuk memungkinkan pemetaan sumber peristiwa ke pesan polling dari klaster, dan memanggil Fungsi Acara dengan menggunakan jalur melalui VPC Anda. Ini diperlukan hanya jika pemetaan sumber acara Anda menggunakan mode sesuai permintaan, dan klaster Anda berada dalam subnet pribadi VPC Anda. Jika klaster Anda berada di subnet publik VPC Anda, atau pemetaan sumber acara Anda menggunakan mode yang disediakan, Anda tidak perlu mengonfigurasi gateway NAT.

Gateway NAT memungkinkan sumber daya dalam subnet pribadi untuk mengakses internet publik. Jika Anda memerlukan konektivitas pribadi ke Lambda, lihat the section called “Melakukan konfigurasi AWS PrivateLink titik akhir untuk sumber acara MSK” saja.

Setelah Anda mengkonfigurasi gateway NAT Anda, Anda harus mengkonfigurasi tabel rute yang sesuai. Ini memungkinkan lalu lintas dari subnet pribadi Anda untuk rute ke internet publik melalui gateway NAT.



Langkah-langkah berikut memandu Anda melalui konfigurasi gateway NAT menggunakan konsol. Ulangi langkah-langkah ini seperlunya untuk setiap Availability Zone (AZ).

Untuk mengonfigurasi gateway NAT dan perutean yang tepat (konsol)

1. Ikuti langkah-langkah di Buat gateway NAT, perhatikan hal berikut:
   - Gateway NAT harus selalu berada di subnet publik. Buat gateway NAT dengan konektivitas publik.
   - Jika kluster MSK Anda direplikasi di beberapa AZ, buat satu gateway NAT per AZ. Misalnya, di setiap AZ, VPC Anda harus memiliki satu subnet pribadi yang berisi cluster Anda, dan satu subnet publik yang berisi gateway NAT Anda. Untuk pengaturan dengan tiga AZ, Anda akan memiliki tiga subnet pribadi, tiga subnet publik, dan tiga gateway NAT.
2. Setelah Anda membuat gateway NAT, buka konsol VPC Amazon dan pilih tabel Route di menu sebelah kiri.

3. Pilih Buat tabel rute.
4. Kaitkan tabel rute ini dengan VPC yang berisi kluster MSK Anda. Secara opsional, masukkan nama untuk tabel rute Anda.
5. Pilih Buat tabel rute.
6. Pilih tabel rute yang baru saja Anda buat.
7. Di bawah tab Asosiasi subnet, pilih Edit asosiasi subnet.
   - Kaitkan tabel rute ini dengan subnet pribadi yang berisi kluster MSK Anda.
8. Pilih Edit rute.
9. Pilih Tambahkan rute:
   1. Untuk Tujuan, pilih `0.0.0.0/0`.
   2. Untuk Target, pilih gateway NAT.
   3. Di kotak pencarian, pilih gateway NAT yang Anda buat di langkah 1. Ini harus menjadi gateway NAT di AZ yang sama dengan subnet pribadi yang berisi cluster MSK Anda (subnet pribadi yang Anda kaitkan dengan tabel rute ini pada langkah 6).
10. Pilih Simpan perubahan.

Melakukan konfigurasi AWS PrivateLink titik akhir untuk sumber acara MSK

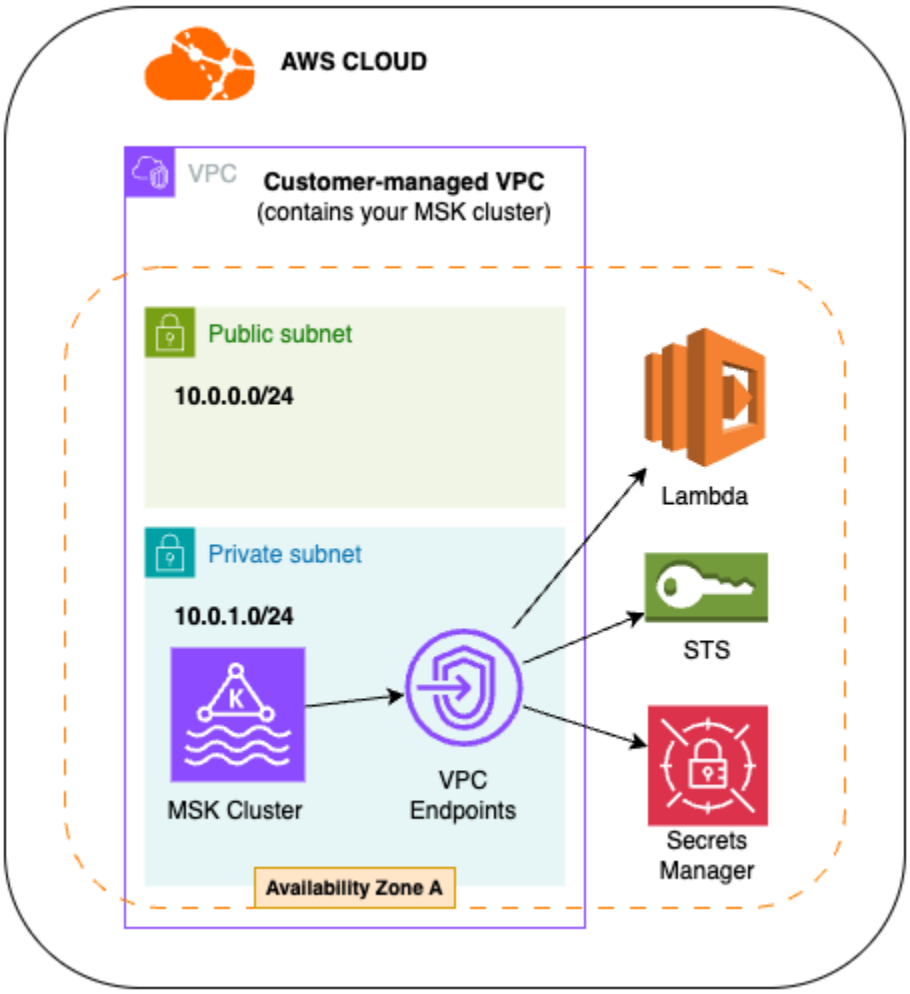
Anda dapat mengonfigurasi AWS PrivateLink titik akhir untuk melakukan polling pesan dari klaster, dan menjalankan fungsi dengan menggunakan jalur melalui VPC Anda. Titik akhir ini akan memungkinkan klaster MSK Anda untuk mengakses hal-hal berikut:

- Layanan Lambda
- AWS Security Token Service (STS)
- Secara opsional, AWS Secrets Managerlayanan. Ini diperlukan jika rahasia yang diperlukan untuk otentikasi cluster disimpan di Secrets Manager.

Mengkonfigurasi PrivateLink titik akhir hanya diperlukan jika pemetaan sumber acara Anda menggunakan mode sesuai permintaan. Jika pemetaan sumber acara Anda menggunakan mode yang disediakan, Lambda membuat koneksi yang diperlukan untuk Anda.

PrivateLink titik akhir memungkinkan akses pribadi yang aman ke AWS layanan berakhir AWS PrivateLink. Atau, untuk mengkonfigurasi gateway NAT untuk memberikan akses kluster MSK Anda ke internet publik, lihat. the section called “Mengkonfigurasi gateway NAT untuk sumber acara MSK”

Setelah Anda mengonfigurasi titik akhir VPC Anda, kluster MSK Anda harus memiliki akses langsung dan pribadi ke Lambda, STS, dan secara opsional, Secrets Manager.



Langkah-langkah berikut memandu Anda dalam mengonfigurasi PrivateLink titik akhir menggunakan konsol. Ulangi langkah-langkah ini seperlunya untuk setiap titik akhir (Lambda, STS, Secrets Manager).

Untuk mengkonfigurasi PrivateLink titik akhir VPC (konsol)

1. Buka konsol Amazon VPC dan pilih Endpoints di menu sebelah kiri.
2. Pilih Buat titik akhir.
3. Secara opsional, masukkan nama untuk titik akhir Anda.
4. Untuk Jenis, pilih AWS layanan.
5. Di bawah Layanan, mulailah mengetik nama layanan. Misalnya, untuk membuat titik akhir untuk terhubung ke Lambda, `lambda` ketik kotak pencarian.

6. Dalam hasilnya, Anda akan melihat titik akhir layanan di wilayah saat ini. Misalnya, di wilayah AS Timur (Virginia N.), Anda harus melihat`com.amazonaws.us-east-2.lambda`. Pilih layanan ini.
7. Di bawah Pengaturan jaringan, pilih VPC yang berisi kluster MSK Anda.
8. Di bawah Subnet, pilih AZ tempat klaster MSK Anda berada.
   - Untuk setiap AZ, di bawah Subnet ID, pilih subnet pribadi yang berisi cluster MSK Anda.
9. Di bawah Grup keamanan, pilih grup keamanan yang terkait dengan kluster MSK Anda.
10. Pilih Buat titik akhir.

Secara default, titik akhir Amazon VPC memiliki kebijakan IAM terbuka yang memungkinkan akses luas ke sumber daya. Praktik terbaik adalah membatasi kebijakan ini untuk melakukan tindakan yang diperlukan menggunakan titik akhir tersebut. Misalnya, untuk titik akhir Secrets Manager, Anda dapat mengubah kebijakannya sehingga hanya mengizinkan peran eksekusi fungsi Anda untuk mengakses rahasia.

Example Kebijakan titik akhir VPC — titik akhir Secrets Manager

```
{
    "Statement": [
        {
            "Action": "secretsmanager:GetSecretValue",
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "arn:aws::iam::123456789012:role/my-role"
                ]
            },
            "Resource": "arn:aws::secretsmanager:us-west-2:123456789012:secret:my-
secret"
        }
    ]
}
```

Untuk titik akhir AWS STS dan Lambda, Anda dapat membatasi prinsipal panggilan ke kepala layanan Lambda. Namun, pastikan Anda menggunakannya `"Resource": "*"` dalam kebijakan ini.

Example Kebijakan titik akhir VPC — AWS STS titik akhir

```
{
    "Statement": [
        {
            "Action": "sts:AssumeRole",
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "lambda.amazonaws.com"
                ]
            },
            "Resource": "*"
        }
    ]
}
```

Example Kebijakan titik akhir VPC - Titik akhir Lambda

```
{
    "Statement": [
        {
            "Action": "lambda:InvokeFunction",
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "lambda.amazonaws.com"
                ]
            },
            "Resource": "*"
        }
    ]
}
```

Mengonfigurasi izin Lambda untuk pemetaan sumber acara MSK Amazon

Untuk mengakses kluster MSK Amazon, Fungsi Acara dan pemetaan sumber peristiwa memerlukan izin untuk melakukan berbagai tindakan Amazon MSK API. Tambahkan izin ini ke peran eksekusi fungsi. Jika pengguna Anda memerlukan akses, tambahkan izin yang diperlukan ke kebijakan identitas untuk pengguna atau peran.

Kebijakan AWSLambdaMSKExecutionRoleterkelola berisi izin minimum yang diperlukan untuk pemetaan sumber peristiwa MSK Lambda Amazon. Untuk menyederhanakan proses izin, Anda dapat:

- Lampirkan kebijakan AWSLambdaMSKExecutionRoleterkelola ke peran eksekusi Anda.
- Biarkan konsol Lambda menghasilkan izin untuk Anda. Saat Anda membuat pemetaan sumber peristiwa MSK Amazon di konsol, Lambda mengevaluasi peran eksekusi Anda dan memberi tahu Anda jika ada izin yang hilang. Pilih Hasilkan izin untuk memperbarui peran eksekusi Anda secara otomatis. Ini tidak berfungsi jika Anda membuat atau memodifikasi kebijakan peran eksekusi secara manual, atau jika kebijakan dilampirkan ke beberapa peran. Perhatikan bahwa izin tambahan mungkin masih diperlukan dalam peran eksekusi Anda saat menggunakan fitur lanjutan seperti On-Failure Destination atau AWS Glue Schema Registry.

Topik

- Izin yang diperlukan
- Izin opsional

Izin yang diperlukan

Peran eksekusi Fungsi Peristiwa Lambda Anda harus memiliki izin yang diperlukan berikut untuk pemetaan sumber peristiwa MSK Amazon. Izin ini disertakan dalam kebijakan terkelola AWSLambdaMSKExecutionRole.

CloudWatch Izin log

Izin berikut memungkinkan Lambda untuk membuat dan menyimpan log di Amazon CloudWatch Logs.

- log: CreateLogGroup
- log: CreateLogStream
- log: PutLogEvents

Izin klaster MSK

Izin berikut memungkinkan Lambda mengakses klaster MSK Amazon Anda atas nama Anda:

- kafka: DescribeCluster

- kafka: DescribeCluster V2
- kafka: GetBootstrapBrokers

Kami merekomendasikan menggunakan kafka: DescribeCluster V2 alih-alih kafka:. DescribeCluster Izin v2 berfungsi dengan kluster MSK Amazon yang disediakan dan tanpa server. Anda hanya memerlukan salah satu izin ini dalam kebijakan Anda.

Izin VPC

Izin berikut memungkinkan Lambda membuat dan mengelola antarmuka jaringan saat menghubungkan ke kluster MSK Amazon Anda:

- EC2: CreateNetworkInterface
- EC2: DescribeNetworkInterfaces
- EC2: DescribeVpcs
- EC2: DeleteNetworkInterface
- EC2: DescribeSubnets
- EC2: DescribeSecurityGroups

Izin opsional

Fungsi Acara Lambda Anda mungkin juga memerlukan izin untuk:

- Akses kluster MSK Amazon lintas akun. Untuk pemetaan sumber peristiwa lintas akun, Anda memerlukan kafka: DescribeVpcConnection dalam peran eksekusi. Prinsipal IAM yang membuat pemetaan sumber peristiwa lintas akun membutuhkan kafka:. ListVpcConnections
- Akses rahasia SCRAM Anda, jika Anda menggunakan SASL/SCRAM otentikasi. Dengan ini, fungsi Anda dapat menggunakan nama pengguna dan kata sandi untuk terhubung ke Kafka.
- Jelaskan rahasia Secrets Manager Anda, jika Anda menggunakan SASL/SCRAM atau otentikasi mTLS. Dengan izin ini, fungsi Anda dapat mengambil kredensyal atau sertifikat yang diperlukan untuk koneksi aman.
- Akses kunci terkelola AWS KMS pelanggan Anda, jika AWS Secrets Manager rahasia Anda dienkripsi dengan kunci yang dikelola AWS KMS pelanggan.
- Akses rahasia registri skema Anda, jika Anda menggunakan registri skema dengan otentikasi:

  - Untuk Registri AWS Glue Skema: Kebutuhan `glue:GetRegistry` dan `glue:GetSchemaVersion` izin fungsi Anda. Ini memungkinkan fungsi Anda untuk mencari dan menggunakan aturan format pesan yang disimpan AWS Glue.
  - Untuk Registri Skema Konfluen dengan `BASIC_AUTH` atau`CLIENT_CERTIFICATE_TLS_AUTH`: Fungsi Anda memerlukan `secretsmanager:GetSecretValue` izin untuk rahasia yang berisi kredensi otentikasi. Ini memungkinkan fungsi Anda mengambil username/password atau sertifikat yang diperlukan untuk mengakses Registri Skema Konfluen.
  - Untuk sertifikat CA pribadi: Fungsi Anda membutuhkan secretsmanager: GetSecretValue izin untuk rahasia yang berisi sertifikat. Dengan izin ini, fungsi Anda dapat memverifikasi identitas pendaftar skema yang menggunakan sertifikat khusus.
- Akses grup konsumen klaster Kafka dan pesan polling dari topik, jika Anda menggunakan autentikasi IAM untuk pemetaan sumber peristiwa.

Ini sesuai dengan izin yang diperlukan berikut:

- kafka: ListScramSecrets - Memungkinkan daftar rahasia SCRAM untuk otentikasi Kafka
- secretsmanager: GetSecretValue - Memungkinkan pengambilan rahasia dari Secrets Manager
- KMS: Decrypt - Memungkinkan dekripsi data terenkripsi menggunakan AWS KMS
- lem: GetRegistry - Memungkinkan akses ke AWS Glue Schema Registry
- lem: GetSchemaVersion - Memungkinkan pengambilan versi skema tertentu dari Schema Registry AWS Glue
- Kafka-cluster:Connect - Memberikan izin untuk menghubungkan dan mengautentikasi ke cluster
- kafka-cluster: AlterGroup - Memberikan izin untuk bergabung dengan grup di cluster, setara dengan READ GROUP ACL Apache Kafka
- kafka-cluster: DescribeGroup - Memberikan izin untuk mendeskripsikan grup di cluster, setara dengan APache Kafka's DESCRIPTION GROUP ACL
- kafka-cluster: DescribeTopic - Memberikan izin untuk mendeskripsikan topik di cluster, setara dengan APache Kafka's DESCRIPTE TOPIC ACL
- kafka-cluster: ReadData - Memberikan izin untuk membaca data dari topik di cluster, setara dengan ACL TOPIK BACA Apache Kafka

Selain itu, jika Anda ingin mengirim catatan pemanggilan yang gagal ke tujuan yang gagal, Anda memerlukan izin berikut tergantung pada jenis tujuan:

- Untuk tujuan Amazon SQS: sqs: SendMessage - Memungkinkan pengiriman pesan ke antrian Amazon SQS
- Untuk tujuan Amazon SNS: SNS: Publikasikan - Izinkan penerbitan pesan ke topik Amazon SNS
- Untuk tujuan bucket Amazon S3: s3: PutObject dan s3: ListBucket - Mengaktifkan penulisan dan daftar objek dalam bucket Amazon S3

Untuk mengatasi masalah kesalahan autentikasi dan otorisasi, lihat. the section called “Pemecahan masalah”

Mengonfigurasi sumber acara MSK Amazon untuk Lambda

Untuk menggunakan kluster MSK Amazon sebagai sumber peristiwa untuk Fungsi Peristiwa Lambda, Anda membuat pemetaan sumber peristiwa yang menghubungkan dua sumber daya. Halaman ini menjelaskan cara membuat pemetaan sumber peristiwa untuk Amazon MSK.

Halaman ini mengasumsikan bahwa Anda telah mengonfigurasi klaster MSK dengan benar dan Amazon Virtual Private Cloud (VPC) tempat ia berada. Jika Anda perlu mengatur cluster atau VPC Anda, lihat. the section called “Pengaturan cluster dan jaringan” Untuk mengonfigurasi perilaku coba lagi untuk penanganan kesalahan, lihatthe section called “Coba lagi konfigurasi”.

Topik

- Menggunakan kluster MSK Amazon sebagai sumber acara
- Mengonfigurasi metode otentikasi klaster MSK Amazon di Lambda
- Membuat pemetaan sumber acara Lambda untuk sumber acara MSK Amazon
- Membuat pemetaan sumber acara lintas akun di Lambda
- Semua parameter konfigurasi sumber acara MSK Amazon di Lambda

Menggunakan kluster MSK Amazon sebagai sumber acara

Saat Anda menambahkan kluster Apache Kafka atau Amazon MSK sebagai pemicu Fungsi Peristiwa Lambda Anda, cluster digunakan sebagai sumber peristiwa.

Lambda membaca data peristiwa dari topik Kafka yang Anda tentukan seperti `Topics` dalam CreateEventSourceMappingpermintaan, berdasarkan posisi awal yang Anda tentukan. Setelah pemrosesan berhasil, topik Kafka Anda dijalankan untuk klaster Kafka Anda.

Lambda membaca pesan secara berurutan untuk setiap partisi topik Kafka. Payload Lambda tunggal dapat berisi pesan dari beberapa partisi. Bila lebih banyak rekaman tersedia, Lambda

terus memproses catatan dalam batch, berdasarkan BatchSize nilai yang Anda tentukan dalam CreateEventSourceMappingpermintaan, hingga fungsi Anda mengikuti topik.

Setelah Lambda memproses setiap batch, Lambda melakukan offset pesan dalam batch tersebut. Jika fungsi Anda mengembalikan kesalahan untuk salah satu pesan dalam batch, Lambda mencoba ulang seluruh batch pesan sampai berhasil diproses atau pesan berakhir. Anda dapat mengirim catatan yang gagal dalam semua upaya percobaan ulang ke tujuan yang gagal untuk diproses nanti.

> **Note**
>
> Sementara Fungsi Acara Lambda biasanya memiliki batas waktu tunggu maksimum 15 menit, pemetaan sumber acara untuk Amazon MSK, Apache Kafka yang dikelola sendiri, Amazon DocumentDB, dan Amazon MQ untuk ActiveMQ dan RabbitMQ hanya mendukung fungsi dengan batas waktu tunggu maksimum 14 menit.

Mengonfigurasi metode otentikasi klaster MSK Amazon di Lambda

Lambda memerlukan izin untuk mengakses kluster MSK Amazon Anda, mengambil catatan, dan melakukan tugas lainnya. Amazon MSK mendukung beberapa cara untuk mengautentikasi dengan kluster MSK Anda.

Metode otentikasi cluster

- Akses tidak diautentikasi
- SASL/SCRAM otentikasi
- Autentikasi TLS bersama
- Autentikasi IAM
- Bagaimana Lambda memilih broker bootstrap

Akses tidak diautentikasi

Jika tidak ada klien yang mengakses cluster melalui internet, Anda dapat menggunakan akses yang tidak diautentikasi.

SASL/SCRAM otentikasi

Lambda mendukung otentikasi Simple Authentication and Security Layer/Salted Challenge Response Authentication Mechanism (SASL/SCRAM) otentikasi, dengan fungsi SHA-512 hash dan enkripsi

Transport Layer Security (TLS). Agar Lambda dapat terhubung ke cluster, simpan kredensyal otentikasi (nama pengguna dan kata sandi) dalam rahasia Secrets Manager, dan rujuk rahasia ini saat mengonfigurasi pemetaan sumber peristiwa Anda.

Untuk informasi selengkapnya tentang menggunakan Secrets Manager, lihat autentikasi Sign-in kredensyal dengan Secrets Manager di Panduan Pengembang Amazon Managed Streaming for Apache Kafka.

> **Note**
>
> Amazon MSK tidak mendukung SASL/PLAIN otentikasi.

Autentikasi TLS bersama

Mutual TLS (mTLS) menyediakan otentikasi dua arah antara klien dan server. Klien mengirimkan sertifikat ke server untuk server untuk memverifikasi klien. Server juga mengirimkan sertifikat ke klien untuk klien untuk memverifikasi server.

Untuk integrasi MSK Amazon dengan Lambda, kluster MSK Anda bertindak sebagai server, dan Lambda bertindak sebagai klien.

- Agar Lambda memverifikasi klaster MSK Anda, Anda mengonfigurasi sertifikat klien sebagai rahasia di Secrets Manager, dan mereferensikan sertifikat ini dalam konfigurasi pemetaan sumber peristiwa Anda. Sertifikat klien harus ditandatangani oleh otoritas sertifikat (CA) di toko kepercayaan server.
- Cluster MSK juga mengirimkan sertifikat server ke Lambda. Sertifikat server harus ditandatangani oleh otoritas sertifikat (CA) di toko AWS kepercayaan.

Amazon MSK tidak mendukung sertifikat server yang ditandatangani sendiri. Semua broker di Amazon MSK menggunakan sertifikat publik yang ditandatangani oleh Amazon Trust Services CA, yang dipercaya Lambda secara default.

Mengkonfigurasi rahasia mTLS

Rahasia CLIENT_CERTIFICATE_TLS_AUTH memerlukan bidang sertifikat dan bidang kunci pribadi. Untuk kunci pribadi terenkripsi, rahasianya memerlukan kata sandi kunci pribadi. Baik sertifikat dan kunci pribadi harus dalam format PEM.

> **Note**
>
> Lambda mendukung algoritma enkripsi kunci pribadi PBES1 (tetapi bukan PBES2).

Bidang sertifikat harus berisi daftar sertifikat, dimulai dengan sertifikat klien, diikuti oleh sertifikat perantara, dan diakhiri dengan sertifikat root. Setiap sertifikat harus dimulai pada baris baru dengan struktur berikut:

```
-----BEGIN CERTIFICATE-----
        <certificate contents>
-----END CERTIFICATE-----
```

Secrets Manager mendukung rahasia hingga 65.536 byte, yang merupakan ruang yang cukup untuk rantai sertifikat yang panjang.

Kunci pribadi harus dalam format PKCS #8, dengan struktur berikut:

```
-----BEGIN PRIVATE KEY-----
         <private key contents>
-----END PRIVATE KEY-----
```

Untuk kunci pribadi terenkripsi, gunakan struktur berikut:

```
-----BEGIN ENCRYPTED PRIVATE KEY-----
          <private key contents>
-----END ENCRYPTED PRIVATE KEY-----
```

Contoh berikut menunjukkan isi rahasia untuk otentikasi mTLS menggunakan kunci pribadi terenkripsi. Untuk kunci pribadi terenkripsi, Anda menyertakan kata sandi kunci pribadi dalam rahasia.

```
{
 "privateKeyPassword": "testpassword",
 "certificate": "-----BEGIN CERTIFICATE-----
MIIE5DCCAsygAwIBAgIRAPJdwaFaNRrytHBto0j5BA0wDQYJKoZIhvcNAQELBQAw
...
j0Lh4/+1HfgyE2KlmII36dg4IMzNjAFEBZiCRoPimO40s1cRqtFHXoal0QQbIlxk
cmUuiAii9R0=
-----END CERTIFICATE-----
```

```
-----BEGIN CERTIFICATE-----
MIIFgjCCA2qgAwIBAgIQdjNZd6uFf9hbNC5RdfmHrzANBgkqhkiG9w0BAQsFADBb
...
rQoiowbbk5wXCheYSANQIfTZ6weQTgiCHCCbuuMKNVS95FkXm0vqVD/YpXKwA/no
c8PH3PSoAaRwMMgOSA2ALJvbRz8mpg==
-----END CERTIFICATE-----",
 "privateKey": "-----BEGIN ENCRYPTED PRIVATE KEY-----
MIIFKzBVBgkqhkiG9w0BBQ0wSDAnBgkqhkiG9w0BBQwwGgQUiAFcK5hT/X7Kjmgp
...
QrSekqF+kWzmB6nAfSzgO9IaoAaytLvNgGTckWeUkWn/V0Ck+LdGUXzAC4RxZnoQ
zp2mwJn2NYB7AZ7+imp0azDZb+8YG2aUCiyqb6PnnA==
-----END ENCRYPTED PRIVATE KEY-----"
}
```

Untuk informasi selengkapnya tentang mTL untuk Amazon MSK, dan petunjuk tentang cara membuat sertifikat klien, lihat Autentikasi klien Mutual TLS untuk Amazon MSK di Panduan Pengembang Amazon Managed Streaming for Apache Kafka.

Autentikasi IAM

Anda dapat menggunakan AWS Identity and Access Management (IAM) untuk mengautentikasi identitas klien yang terhubung ke cluster MSK. Dengan autentikasi IAM, Lambda bergantung pada izin dalam peran eksekusi Fungsi Acara Anda untuk terhubung ke klaster, mengambil catatan, dan melakukan tindakan lain yang diperlukan. Untuk contoh kebijakan yang berisi izin yang diperlukan, lihat Membuat kebijakan otorisasi untuk peran IAM di Panduan Pengembang Amazon Managed Streaming for Apache Kafka.

Jika autentikasi IAM aktif di kluster MSK Anda, dan Anda tidak memberikan rahasia, Lambda secara otomatis default menggunakan autentikasi IAM.

Untuk informasi selengkapnya tentang autentikasi IAM di Amazon MSK, lihat kontrol akses IAM.

Bagaimana Lambda memilih broker bootstrap

Lambda memilih broker bootstrap berdasarkan metode otentikasi yang tersedia di cluster Anda, dan apakah Anda memberikan rahasia untuk otentikasi. Jika Anda memberikan rahasia untuk mTL atau SASL/SCRAM, Lambda secara otomatis memilih metode autentikasi itu. Jika Anda tidak memberikan rahasia, Lambda memilih metode autentikasi terkuat yang aktif di cluster Anda. Berikut ini adalah urutan prioritas di mana Lambda memilih broker, dari autentikasi terkuat hingga terlemah:

- mTL (rahasia disediakan untuk mTL)

- SASL/SCRAM (rahasia disediakan untuk SASL/SCRAM)
- SASL IAM (tidak ada rahasia yang disediakan, dan autentikasi IAM aktif)
- TLS yang tidak diautentikasi (tidak ada rahasia yang disediakan, dan autentikasi IAM tidak aktif)
- Plaintext (tidak ada rahasia yang disediakan, dan autentikasi IAM dan TLS yang tidak diautentikasi tidak aktif)

> **Note**
>
> Jika Lambda tidak dapat terhubung ke jenis broker yang paling aman, Lambda tidak mencoba untuk terhubung ke jenis broker yang berbeda (lebih lemah). Jika Anda ingin Lambda memilih jenis broker yang lebih lemah, nonaktifkan semua metode autentikasi yang lebih kuat di cluster Anda.

Membuat pemetaan sumber acara Lambda untuk sumber acara MSK Amazon

Untuk membuat pemetaan sumber peristiwa, Anda dapat menggunakan konsol Lambda, AWS Command Line Interface (CLI), atau SDK.AWS

> **Note**
>
> Saat Anda membuat pemetaan sumber peristiwa, Lambda membuat ENI hyperplane di subnet pribadi yang berisi kluster MSK Anda, memungkinkan Lambda membuat koneksi aman. ENI hyperplane ini memungkinkan penggunaan subnet dan konfigurasi grup keamanan klaster MSK Anda, bukan Fungsi Acara Lambda Anda.

Langkah-langkah konsol berikut menambahkan kluster MSK Amazon sebagai pemicu Fungsi Peristiwa Lambda Anda. Di bawah tenda, ini menciptakan sumber daya pemetaan sumber peristiwa.

Untuk menambahkan pemicu MSK Amazon ke Fungsi Peristiwa Lambda (konsol)

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih nama Fungsi Peristiwa Lambda yang ingin Anda tambahkan pemicu MSK Amazon.
3. Di bagian Gambaran umum fungsi, pilih Tambah pemicu.
4. Di bawah konfigurasi Trigger, pilih MSK.
5. Untuk menentukan detail cluster Kafka Anda, lakukan hal berikut:

1. Untuk Klaster MSK, pilih klaster Anda.
2. Untuk nama Topik, masukkan nama topik Kafka untuk mengkonsumsi pesan dari.
3. Untuk ID grup Konsumen, masukkan ID grup konsumen Kafka untuk bergabung, jika berlaku. Untuk informasi selengkapnya, lihat the section called “ID grup konsumen”.

6. Untuk otentikasi Cluster, buat konfigurasi yang diperlukan. Untuk informasi selengkapnya tentang otentikasi klaster, lihatthe section called “Autentikasi klaster”.

   - Aktifkan Gunakan otentikasi jika Anda ingin Lambda melakukan autentikasi dengan klaster MSK Anda saat membuat koneksi. Otentikasi direkomendasikan.
   - Jika Anda menggunakan otentikasi, untuk metode otentikasi, pilih metode otentikasi yang akan digunakan.
   - Jika Anda menggunakan autentikasi, untuk kunci Secrets Manager, pilih kunci Secrets Manager yang berisi kredensyal otentikasi yang diperlukan untuk mengakses klaster Anda.

7. Di bawah konfigurasi Event poller, buat konfigurasi yang diperlukan.

   - Pilih Aktifkan pemicu untuk mengaktifkan pemicu segera setelah pembuatan.
   - Pilih apakah Anda ingin Mengonfigurasi mode yang disediakan untuk pemetaan sumber acara Anda. Untuk informasi selengkapnya, lihat the section called “Penskalaan poller acara”.
     - Jika Anda mengonfigurasi mode yang disediakan, masukkan nilai untuk poller peristiwa Minimum, nilai untuk poller peristiwa maksimum, dan nilai opsional PollerGroupName untuk menentukan pengelompokan beberapa ESM dalam VPC sumber peristiwa yang sama.
   - Untuk posisi awal, pilih bagaimana Anda ingin Lambda mulai membaca dari streaming Anda. Untuk informasi selengkapnya, lihat the section called “Posisi polling dan streaming”.

8. Di bawah Batching, buat konfigurasi yang diperlukan. Untuk informasi lebih lanjut tentang batching, lihatthe section called “Perilaku batching”.

   1. Untuk Ukuran batch, masukkan jumlah maksimum pesan yang akan diterima dalam satu batch.
   2. Untuk jendela Batch, masukkan jumlah detik maksimum yang dihabiskan Lambda untuk mengumpulkan catatan sebelum menjalankan fungsi.

9. Di bawah Pemfilteran, buat konfigurasi yang diperlukan. Untuk informasi selengkapnya tentang pemfilteran, lihat the section called “Pemfilteran peristiwa”.

   - Untuk kriteria Filter, tambahkan definisi kriteria filter untuk menentukan apakah akan memproses suatu peristiwa atau tidak.

10. Di bawah Penanganan kegagalan, buat konfigurasi yang diperlukan. Untuk informasi selengkapnya tentang penanganan kegagalan, lihatthe section called “Pertahankan pemanggilan yang gagal”.
    - Untuk On-failure tujuan, tentukan ARN tujuan kegagalan Anda.
11. Untuk Tag, masukkan tag untuk dikaitkan dengan pemetaan sumber peristiwa ini.
12. Untuk membuat pemicu, pilih Tambahkan.

Anda juga dapat membuat pemetaan sumber peristiwa menggunakan CLI dengan perintah AWS create-event-source-mapping. Contoh berikut membuat pemetaan sumber peristiwa untuk memetakan `my-msk-function` Fungsi Peristiwa Lambda ke `AWSKafkaTopic` topik, mulai dari `LATEST` pesan. Perintah ini juga menggunakan SourceAccessConfigurationobjek untuk menginstruksikan Lambda untuk SASL/SCRAMmenggunakan otentikasi saat menghubungkan ke cluster.

```
aws lambda create-event-source-mapping \
  --event-source-arn arn:aws:kafka:us-east-1:111122223333:cluster/my-cluster/fc2f5bdf-
fd1b-45ad-85dd-15b4a5a6247e-2 \
  --topics AWSKafkaTopic \
  --starting-position LATEST \
  --function-name my-kafka-function
  --source-access-configurations '[{"Type": "SASL_SCRAM_512_AUTH","URI":
 "arn:aws:secretsmanager:us-east-1:111122223333:secret:my-secret"}]'
```

Jika cluster menggunakan otentikasi mTLS, sertakan SourceAccessConfigurationobjek yang menentukan `CLIENT_CERTIFICATE_TLS_AUTH` dan ARN kunci Secrets Manager. Ini ditunjukkan dalam perintah berikut:

```
aws lambda create-event-source-mapping \
  --event-source-arn arn:aws:kafka:us-east-1:111122223333:cluster/my-cluster/fc2f5bdf-
fd1b-45ad-85dd-15b4a5a6247e-2 \
  --topics AWSKafkaTopic \
  --starting-position LATEST \
  --function-name my-kafka-function
  --source-access-configurations '[{"Type": "CLIENT_CERTIFICATE_TLS_AUTH","URI":
 "arn:aws:secretsmanager:us-east-1:111122223333:secret:my-secret"}]'
```

Saat cluster menggunakan otentikasi IAM, Anda tidak memerlukan SourceAccessConfigurationobjek. Ini ditunjukkan dalam perintah berikut:

```
aws lambda create-event-source-mapping \
  --event-source-arn arn:aws:kafka:us-east-1:111122223333:cluster/my-cluster/fc2f5bdf-
fd1b-45ad-85dd-15b4a5a6247e-2 \
  --topics AWSKafkaTopic \
  --starting-position LATEST \
  --function-name my-kafka-function
```

Membuat pemetaan sumber acara lintas akun di Lambda

Anda dapat menggunakan konektivitas pribadi multi-VPC untuk menghubungkan Fungsi Peristiwa Lambda ke kluster MSK yang disediakan secara berbeda. Akun AWS Multi-VPC penggunaan konektivitas AWS PrivateLink, yang menjaga semua lalu lintas dalam AWS jaringan.

> **Note**
>
> Anda tidak dapat membuat pemetaan sumber peristiwa lintas akun untuk kluster MSK tanpa server.

Untuk membuat pemetaan sumber peristiwa lintas akun, Anda harus terlebih dahulu mengonfigurasi konektivitas multi-VPC untuk kluster MSK. Saat Anda membuat pemetaan sumber peristiwa, gunakan ARN koneksi VPC terkelola alih-alih ARN cluster, seperti yang ditunjukkan pada contoh berikut. CreateEventSourceMappingOperasi juga berbeda tergantung pada jenis otentikasi yang digunakan kluster MSK.

Example— Buat pemetaan sumber peristiwa lintas akun untuk cluster yang menggunakan otentikasi IAM

Saat cluster menggunakan autentikasi berbasis peran IAM, Anda tidak memerlukan objek. SourceAccessConfiguration Contoh:

```
aws lambda create-event-source-mapping \
  --event-source-arn arn:aws:kafka:us-east-1:111122223333:vpc-connection/444455556666/
my-cluster-name/51jn98b4-0a61-46cc-b0a6-61g9a3d797d5-7 \
  --topics AWSKafkaTopic \
  --starting-position LATEST \
  --function-name my-kafka-function
```

Example— Buat pemetaan sumber peristiwa lintas akun untuk cluster yang menggunakan otentikasi SASL/SCRAM

Jika cluster menggunakan SASL/SCRAM otentikasi, Anda harus menyertakan SourceAccessConfigurationobjek yang menentukan `SASL_SCRAM_512_AUTH` dan Rahasia Secrets Manager ARN.

Ada dua cara untuk menggunakan rahasia untuk pemetaan sumber peristiwa Amazon MSK lintas akun dengan otentikasi: SASL/SCRAM

- Buat rahasia di akun Lambda Event Function dan sinkronkan dengan rahasia cluster. Buat rotasi untuk menjaga kedua rahasia tetap sinkron. Opsi ini memberi Anda kendali atas rahasia dari akun fungsi.
- Gunakan rahasia yang terkait dengan cluster MSK. Rahasia ini harus memungkinkan akses lintas akun ke akun Fungsi Acara Lambda. Untuk informasi selengkapnya, lihat Izin untuk AWS Secrets Manager rahasia bagi pengguna di akun lain.

```
aws lambda create-event-source-mapping \
  --event-source-arn arn:aws:kafka:us-east-1:111122223333:vpc-connection/444455556666/
my-cluster-name/51jn98b4-0a61-46cc-b0a6-61g9a3d797d5-7 \
  --topics AWSKafkaTopic \
  --starting-position LATEST \
  --function-name my-kafka-function \
  --source-access-configurations '[{"Type": "SASL_SCRAM_512_AUTH","URI":
 "arn:aws:secretsmanager:us-east-1:444455556666:secret:my-secret"}]'
```

Example— Buat pemetaan sumber peristiwa lintas akun untuk cluster yang menggunakan otentikasi mTLS

Jika cluster menggunakan otentikasi mTLS, Anda harus menyertakan SourceAccessConfigurationobjek yang menentukan `CLIENT_CERTIFICATE_TLS_AUTH` dan ARN rahasia Secrets Manager. Rahasianya dapat disimpan di akun cluster atau akun Fungsi Acara Lambda.

```
aws lambda create-event-source-mapping \
  --event-source-arn arn:aws:kafka:us-east-1:111122223333:vpc-connection/444455556666/
my-cluster-name/51jn98b4-0a61-46cc-b0a6-61g9a3d797d5-7 \
  --topics AWSKafkaTopic \
  --starting-position LATEST \
  --function-name my-kafka-function \
```

```
  --source-access-configurations '[{"Type": "CLIENT_CERTIFICATE_TLS_AUTH","URI":
"arn:aws:secretsmanager:us-east-1:444455556666:secret:my-secret"}]'
```

Semua parameter konfigurasi sumber acara MSK Amazon di Lambda

Semua jenis sumber peristiwa Lambda berbagi operasi yang sama CreateEventSourceMappingdan UpdateEventSourceMappingAPI. Namun, hanya beberapa parameter yang berlaku untuk Amazon MSK, seperti yang ditunjukkan pada tabel berikut.

Parameter	Diperlukan	Default	Catatan
AmazonManagedKafkaEventSourceConfig	T	Berisi ConsumerGroupId bidang, yang default ke nilai unik.	Hanya dapat mengatur di Create
BatchSize	T	100	Maksimum: 10.000.
DestinationConfig	T	N/A	the section called “Pertahankan pemanggilan yang gagal”
Diaktifkan	T	True	
BisectBatchOnFunctionError	T	False	the section called “Coba lagi konfigurasi”
FunctionResponseTypes	T	N/A	the section called “Coba lagi konfigurasi”
MaximumRecordAgeInSeconds	T	-1 (tak terbatas)	the section called “Coba lagi konfigurasi”
MaximumRetryAttempts	T	-1 (tak terbatas)	the section called “Coba lagi konfigurasi”

Parameter	Diperlukan	Default	Catatan
EventSourceArn	Y	N/A	Hanya dapat mengatur di Create
FilterCriteria	T	N/A	Kontrol peristiwa mana yang dikirim Lambda ke Fungsi Acara Anda
FunctionName	Y	N/A	
KMSKeyArn	T	N/A	the section called “Enkripsi kriteria filter”
MaximumBatchingWindowInSeconds	T	500 ms	Perilaku batching
ProvisionedPollersConfig	T	`MinimumPollers` : nilai default 1 jika tidak ditentukan `MaximumPollers` : nilai default 200 jika tidak ditentukan `PollerGroupName` : N/A	the section called “Mode yang disediakan”
SourceAccessConfigurations	T	Tidak ada kredensial	SASL/SCRAM atau kredensyal otentikasi CLIENT_CERTIFICATE _TLS_AUTH (MutualTLS) untuk sumber acara Anda

Parameter	Diperlukan	Default	Catatan
StartingPosition	Y	N/A	AT_TIMESTAMP, TRIM_HORIZON, atau TERBARU Hanya dapat mengatur di Create
StartingPositionTimestamp	T	N/A	Diperlukan jika StartingPosition disetel ke AT_TIMESTAMP
Tanda	T	N/A	the section called “Tag pemetaan sumber acara”
Topik	Y	N/A	Nama topik Kafka Hanya dapat mengatur di Create

Note

Saat Anda menentukan`PollerGroupName`, beberapa ESM dalam VPC Amazon yang sama dapat berbagi kapasitas Event Poller Unit (EPU). Anda dapat menggunakan opsi ini untuk mengoptimalkan biaya Mode yang disediakan untuk ESM Anda. Persyaratan untuk pengelompokan ESM:

- ESM harus berada dalam VPC Amazon yang sama
- Maksimal 100 ESM per kelompok poller
- Agregat poller maksimum di semua ESM dalam grup tidak boleh melebihi 2000

Anda dapat memperbarui `PollerGroupName` untuk memindahkan ESM ke grup lain, atau menghapus ESM dari grup dengan menyetel `PollerGroupName` ke string kosong (“”).

Tutorial: Menggunakan pemetaan sumber acara MSK Amazon untuk menjalankan Fungsi Acara Lambda

Dalam tutorial ini, Anda melakukan hal berikut:

- Buat Fungsi Acara Lambda di AWS akun yang sama dengan kluster MSK Amazon yang ada.
- Konfigurasikan jaringan dan otentikasi untuk Lambda untuk berkomunikasi dengan Amazon MSK.
- Siapkan pemetaan sumber peristiwa MSK Lambda Amazon, yang menjalankan fungsi Anda saat peristiwa muncul di topik.

Setelah Anda selesai dengan langkah-langkah ini, ketika acara dikirim ke Amazon MSK, Anda dapat mengatur fungsi untuk memproses peristiwa tersebut secara otomatis dengan kode Lambda kustom Anda sendiri.

Apa yang dapat Anda lakukan dengan fitur ini?

Contoh solusi: Gunakan pemetaan sumber acara MSK untuk memberikan skor langsung kepada pelanggan Anda.

Pertimbangkan skenario berikut: Perusahaan Anda menghosting aplikasi web tempat pelanggan Anda dapat melihat informasi tentang acara langsung, seperti permainan olahraga. Pembaruan informasi dari game disediakan untuk tim Anda melalui topik Kafka di Amazon MSK. Anda ingin merancang solusi yang menggunakan pembaruan dari topik MSK untuk memberikan tampilan terbaru dari acara langsung kepada pelanggan di dalam aplikasi yang Anda kembangkan. Anda telah memutuskan pendekatan desain berikut: Aplikasi klien Anda berkomunikasi dengan backend tanpa server yang dihosting di. AWS Klien terhubung melalui sesi websocket menggunakan Amazon API Gateway WebSocket API.

Dalam solusi ini, Anda memerlukan komponen yang membaca peristiwa MSK, melakukan beberapa logika khusus untuk mempersiapkan peristiwa tersebut untuk lapisan aplikasi dan kemudian meneruskan informasi tersebut ke API Gateway API. Anda dapat mengimplementasikan komponen ini dengan AWS Lambda, dengan menyediakan logika kustom Anda dalam suatu fungsi, lalu memanggilnya dengan pemetaan sumber peristiwa MSK AWS Lambda Amazon.

Untuk informasi selengkapnya tentang penerapan solusi menggunakan Amazon API Gateway WebSocket API, lihat tutorial WebSocket API di dokumentasi API Gateway.

Prasyarat

AWS Akun dengan sumber daya yang telah dikonfigurasikan berikut:

Untuk memenuhi prasyarat ini, kami sarankan mengikuti Memulai menggunakan Amazon MSK di dokumentasi MSK Amazon.

- Cluster MSK Amazon. Lihat Membuat klaster MSK Amazon di Memulai menggunakan Amazon MSK.
- Konfigurasi berikut:
  - Pastikan autentikasi berbasis peran IAM Diaktifkan di pengaturan keamanan klaster Anda. Ini meningkatkan keamanan Anda dengan membatasi Fungsi Peristiwa Lambda Anda untuk hanya mengakses sumber daya MSK Amazon yang diperlukan. Ini diaktifkan secara default pada kluster MSK Amazon baru.
  - Pastikan akses Publik tidak aktif di pengaturan jaringan cluster Anda. Membatasi akses klaster MSK Amazon Anda ke internet meningkatkan keamanan Anda dengan membatasi berapa banyak perantara yang menangani data Anda. Ini diaktifkan secara default pada kluster MSK Amazon baru.
- Topik Kafka di cluster MSK Amazon Anda untuk digunakan untuk solusi ini. Lihat Membuat topik di Memulai menggunakan Amazon MSK.
- Host admin Kafka disiapkan untuk mengambil informasi dari cluster Kafka Anda dan mengirim acara Kafka ke topik Anda untuk pengujian, seperti instans Amazon EC2 dengan CLI admin Kafka dan perpustakaan Amazon MSK IAM diinstal. Lihat Membuat mesin klien di Memulai menggunakan Amazon MSK.

Setelah Anda menyiapkan sumber daya ini, kumpulkan informasi berikut dari AWS akun Anda untuk mengonfirmasi bahwa Anda siap untuk melanjutkan.

- Nama cluster MSK Amazon Anda. Anda dapat menemukan informasi ini di konsol MSK Amazon.
- Cluster UUID, bagian dari ARN untuk cluster MSK Amazon Anda, yang dapat Anda temukan di konsol MSK Amazon. Ikuti prosedur di Daftar klaster di dokumentasi MSK Amazon untuk menemukan informasi ini.
- Grup keamanan yang terkait dengan kluster MSK Amazon Anda. Anda dapat menemukan informasi ini di konsol MSK Amazon. Dalam langkah-langkah berikut, lihat ini sebagai milik Anda`clusterSecurityGroups`.
- Id dari VPC Amazon yang berisi kluster MSK Amazon Anda. Anda dapat menemukan informasi ini dengan mengidentifikasi subnet yang terkait dengan kluster MSK Amazon Anda di konsol MSK Amazon, lalu mengidentifikasi VPC Amazon yang terkait dengan subnet di Konsol VPC Amazon.

- Nama topik Kafka yang digunakan dalam solusi Anda. Anda dapat menemukan informasi ini dengan menghubungi cluster MSK Amazon Anda dengan Kafka `topics` CLI dari host admin Kafka Anda. Untuk informasi selengkapnya tentang topik CLI, lihat Menambahkan dan menghapus topik dalam dokumentasi Kafka.
- Nama grup konsumen untuk topik Kafka Anda, cocok untuk digunakan oleh Fungsi Acara Lambda Anda. Grup ini dapat dibuat secara otomatis oleh Lambda, jadi Anda tidak perlu membuatnya dengan Kafka CLI. Jika Anda perlu mengelola grup konsumen Anda, untuk mempelajari lebih lanjut tentang CLI kelompok konsumen, lihat Mengelola Grup Konsumen di dokumentasi Kafka.

Izin berikut di AWS akun Anda:

- Izin untuk membuat dan mengelola fungsi.
- Izin untuk membuat kebijakan IAM dan mengaitkannya dengan fungsi Anda.
- Izin untuk membuat titik akhir VPC Amazon dan mengubah konfigurasi jaringan di VPC Amazon yang menghosting kluster MSK Amazon Anda.

Instal AWS Command Line Interface

Jika Anda belum menginstal AWS Command Line Interface, ikuti langkah-langkah di Menginstal atau memperbarui versi terbaru AWS CLI untuk menginstalnya.

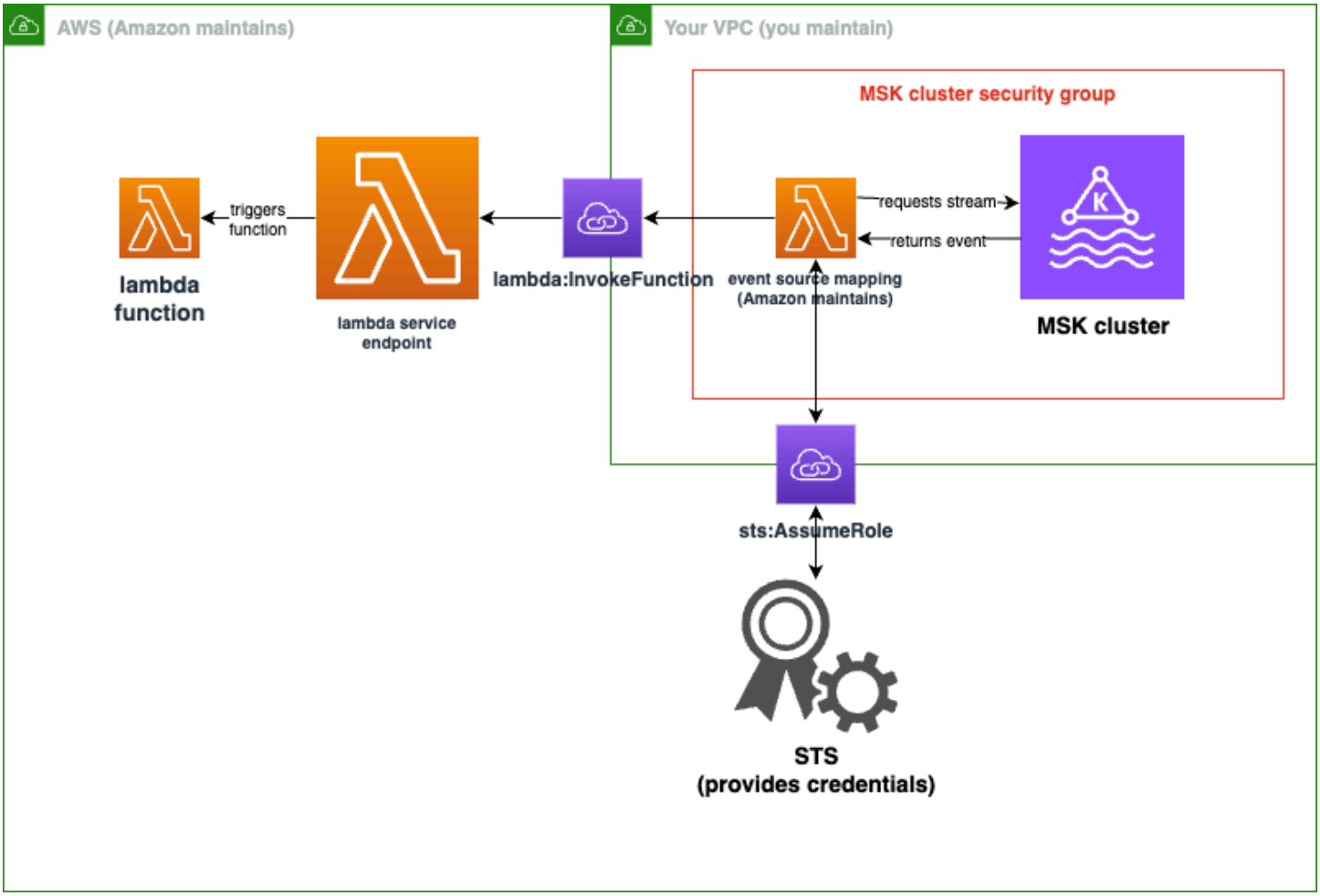
Tutorial ini membutuhkan terminal baris perintah atau shell untuk menjalankan perintah. Di Linux dan macOS, gunakan shell dan manajer paket pilihan Anda.

> **Note**
>
> Di Windows, beberapa perintah Bash CLI yang biasa Anda gunakan dengan Lambda (`zip`seperti) tidak didukung oleh terminal bawaan sistem operasi. Untuk mendapatkan Windows-integrated versi Ubuntu dan Bash, instal Subsistem Windows untuk Linux.

Konfigurasikan konektivitas jaringan untuk Lambda untuk berkomunikasi dengan Amazon MSK

Gunakan AWS PrivateLink untuk menghubungkan Lambda dan Amazon MSK. Anda dapat melakukannya dengan membuat antarmuka titik akhir VPC Amazon di konsol VPC Amazon. Untuk informasi selengkapnya tentang konfigurasi jaringan, lihatthe section called “Pengaturan cluster dan jaringan”.

Saat pemetaan sumber peristiwa MSK Amazon berjalan atas nama Fungsi Peristiwa Lambda, pemetaan tersebut akan mengasumsikan peran eksekusi fungsi tersebut. Peran IAM ini mengizinkan pemetaan untuk mengakses sumber daya yang diamankan oleh IAM, seperti kluster MSK Amazon Anda. Meskipun komponen berbagi peran eksekusi, pemetaan MSK Amazon dan fungsi Anda memiliki persyaratan konektivitas terpisah untuk tugasnya masing-masing, seperti yang ditunjukkan pada diagram berikut.



Pemetaan sumber acara Anda milik grup keamanan kluster MSK Amazon Anda. Pada langkah jaringan ini, buat titik akhir VPC Amazon dari VPC cluster MSK Amazon Anda untuk menghubungkan pemetaan sumber peristiwa ke layanan Lambda dan STS. Amankan titik akhir ini untuk menerima lalu lintas dari grup keamanan klaster MSK Amazon Anda. Kemudian, sesuaikan grup keamanan klaster MSK Amazon untuk memungkinkan pemetaan sumber peristiwa berkomunikasi dengan cluster MSK Amazon.

Anda dapat mengonfigurasi langkah-langkah berikut menggunakan file Konsol Manajemen AWS.

Untuk mengonfigurasi antarmuka titik akhir Amazon VPC untuk menghubungkan Lambda dan Amazon MSK

1. Buat grup keamanan untuk antarmuka Anda titik akhir Amazon VPC,*endpointSecurityGroup*, yang memungkinkan lalu lintas TCP masuk pada 443 dari. *clusterSecurityGroups* Ikuti prosedur di Buat grup keamanan di dokumentasi Amazon EC2 untuk membuat grup keamanan. Kemudian, ikuti prosedur di Tambahkan aturan ke grup keamanan di dokumentasi Amazon EC2 untuk menambahkan aturan yang sesuai.

   Buat grup keamanan dengan informasi berikut:

   Saat menambahkan aturan masuk Anda, buat aturan untuk setiap grup keamanan di*clusterSecurityGroups*. Untuk setiap aturan:

   - Untuk Type, pilih HTTPS.
   - Untuk Sumber, pilih salah satu dari*clusterSecurityGroups*.
2. Buat titik akhir yang menghubungkan layanan Lambda ke VPC Amazon yang berisi kluster MSK Amazon Anda. Ikuti prosedur di Buat titik akhir antarmuka.

   Buat titik akhir antarmuka dengan informasi berikut:

   - Untuk nama Layanan, pilih`com.amazonaws.`*regionName*`.lambda`, di mana *regionName* host fungsi Anda.
   - Untuk VPC, pilih VPC Amazon yang berisi kluster MSK Amazon Anda.
   - Untuk grup Keamanan, pilih*endpointSecurityGroup*, yang Anda buat sebelumnya.
   - Untuk Subnet, pilih subnet yang meng-host cluster MSK Amazon Anda.
   - Untuk Kebijakan, berikan dokumen kebijakan berikut, yang mengamankan titik akhir untuk digunakan oleh kepala layanan Lambda untuk tindakan tersebut. `lambda:InvokeFunction`

```
{
    "Statement": [
        {
            "Action": "lambda:InvokeFunction",
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "lambda.amazonaws.com"
                ]
            },
```

```
            "Resource": "*"
        }
    ]
}
```

- Pastikan Aktifkan nama DNS tetap disetel.

3. Buat titik akhir yang menghubungkan AWS STS layanan ke VPC Amazon yang berisi kluster MSK Amazon Anda. Ikuti prosedur di Buat titik akhir antarmuka.

   Buat titik akhir antarmuka dengan informasi berikut:

   - Untuk nama Layanan, pilih AWS STS.
   - Untuk VPC, pilih VPC Amazon yang berisi kluster MSK Amazon Anda.
   - Untuk grup Keamanan, pilih*endpointSecurityGroup*.
   - Untuk Subnet, pilih subnet yang meng-host cluster MSK Amazon Anda.
   - Untuk Kebijakan, berikan dokumen kebijakan berikut, yang mengamankan titik akhir untuk digunakan oleh kepala layanan Lambda untuk tindakan tersebut. `sts:AssumeRole`

   ```
   {
       "Statement": [
           {
               "Action": "sts:AssumeRole",
               "Effect": "Allow",
               "Principal": {
                   "Service": [
                       "lambda.amazonaws.com"
                   ]
               },
               "Resource": "*"
           }
       ]
   }
   ```

   - Pastikan Aktifkan nama DNS tetap disetel.

4. Untuk setiap grup keamanan yang terkait dengan kluster MSK Amazon Anda, yaitu*clusterSecurityGroups*, izinkan hal berikut:

   - Izinkan semua lalu lintas TCP masuk dan keluar pada 9098 ke semua*clusterSecurityGroups*, termasuk di dalam dirinya sendiri.
   - Izinkan semua lalu lintas TCP keluar pada 443.

Beberapa lalu lintas ini diizinkan oleh aturan grup keamanan default, jadi jika klaster Anda dilampirkan ke satu grup keamanan, dan grup tersebut memiliki aturan default, aturan tambahan tidak diperlukan. Untuk menyesuaikan aturan grup keamanan, ikuti prosedur di Tambahkan aturan ke grup keamanan di dokumentasi Amazon EC2.

Tambahkan aturan ke grup keamanan Anda dengan informasi berikut:

- Untuk setiap aturan masuk atau aturan keluar untuk port 9098, sediakan
  - Untuk Jenis, pilih TCP Kustom.
  - Untuk jangkauan Port, sediakan 9098.
  - Untuk Sumber, berikan salah satu dari*clusterSecurityGroups*.
- Untuk setiap aturan masuk untuk port 443, untuk Type, pilih HTTPS.

Buat peran IAM untuk dibaca Lambda dari topik MSK Amazon Anda

Identifikasi persyaratan autentikasi untuk dibaca Lambda dari topik MSK Amazon Anda, lalu tentukan dalam kebijakan. Buat peran,*lambdaAuthRole*, yang mengizinkan Lambda untuk menggunakan izin tersebut. Otorisasi tindakan di klaster MSK Amazon Anda menggunakan tindakan `kafka-cluster` IAM. Kemudian, beri wewenang kepada Lambda untuk melakukan tindakan Amazon `kafka` MSK dan Amazon EC2 yang diperlukan untuk menemukan dan terhubung ke kluster MSK Amazon Anda, serta CloudWatch tindakan agar Lambda dapat mencatat apa yang telah dilakukannya.

Untuk menjelaskan persyaratan autentikasi agar Lambda dapat dibaca dari Amazon MSK

1. Tulis dokumen kebijakan IAM (dokumen JSON),*clusterAuthPolicy*, yang memungkinkan Lambda membaca dari topik Kafka Anda di kluster MSK Amazon Anda menggunakan grup konsumen Kafka Anda. Lambda membutuhkan kelompok konsumen Kafka untuk diatur saat membaca.

   Ubah template berikut agar selaras dengan prasyarat Anda:

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Action": [
                "kafka-cluster:Connect",
                "kafka-cluster:DescribeGroup",
                "kafka-cluster:AlterGroup",
                "kafka-cluster:DescribeTopic",
                "kafka-cluster:ReadData",
                "kafka-cluster:DescribeClusterDynamicConfiguration"
            ],
            "Resource": [
                "arn:aws:kafka:us-
east-1:111122223333:cluster/mskClusterName/cluster-uuid",
                "arn:aws:kafka:us-
east-1:111122223333:topic/mskClusterName/cluster-uuid/mskTopicName",
                "arn:aws:kafka:us-
east-1:111122223333:group/mskClusterName/cluster-uuid/mskGroupName"
            ]
        }
    ]
}
```

Untuk informasi lebih lanjut, konsultasikanthe section called “Mengatur Konfigurasi Izin ”. Saat menulis kebijakan Anda:

- Ganti *us-east-1* dan *111122223333* dengan Wilayah AWS dan Akun AWS dari kluster MSK Amazon Anda.
- Untuk*mskClusterName*, berikan nama cluster MSK Amazon Anda.
- Untuk*cluster-uuid*, berikan UUID di ARN untuk cluster MSK Amazon Anda.
- Untuk*mskTopicName*, berikan nama topik Kafka Anda.
- Untuk*mskGroupName*, berikan nama grup konsumen Kafka Anda.

2. Identifikasi MSK Amazon, Amazon EC2, CloudWatch dan izin yang diperlukan Lambda untuk menemukan dan menghubungkan kluster MSK Amazon Anda, dan mencatat peristiwa tersebut.

   Kebijakan `AWSLambdaMSKExecutionRole` terkelola secara permisif mendefinisikan izin yang diperlukan. Gunakan dalam langkah-langkah berikut.

Dalam lingkungan produksi, nilai `AWSLambdaMSKExecutionRole` untuk membatasi kebijakan peran eksekusi Anda berdasarkan prinsip hak istimewa terkecil, lalu tulis kebijakan untuk peran Anda yang menggantikan kebijakan terkelola ini.

Untuk detail tentang bahasa kebijakan IAM, lihat dokumentasi IAM.

Sekarang setelah Anda menulis dokumen kebijakan Anda, buat kebijakan IAM sehingga Anda dapat melampirkannya ke peran Anda. Anda dapat melakukan ini menggunakan konsol dengan prosedur berikut.

Untuk membuat kebijakan IAM dari dokumen kebijakan Anda

1. Masuk ke Konsol Manajemen AWS dan buka konsol IAM di https://console.aws.amazon.com/iam/.
2. Di panel navigasi sebelah kiri, pilih Kebijakan.
3. Pilih Buat kebijakan.
4. Di bagian Editor kebijakan, pilih opsi JSON.
5. Tempel*`clusterAuthPolicy`*.
6. Setelah selesai menambahkan izin ke kebijakan, pilih Berikutnya.
7. Pada halaman Tinjau dan buat, ketik Nama Kebijakan dan Deskripsi (opsional) untuk kebijakan yang Anda buat. Tinjau Izin yang ditentukan dalam kebijakan ini untuk melihat izin yang diberikan oleh kebijakan Anda.
8. Pilih Buat kebijakan untuk menyimpan kebijakan baru Anda.

Untuk informasi selengkapnya, lihat Membuat kebijakan IAM di dokumentasi IAM.

Sekarang setelah Anda memiliki kebijakan IAM yang sesuai, buat peran dan lampirkan padanya. Anda dapat melakukan ini menggunakan konsol dengan prosedur berikut.

Untuk membuat peran eksekusi di konsol IAM

1. Buka Halaman peran di konsol IAM.
2. Pilih Buat peran.
3. Di bawah Jenis entitas tepercaya, pilih AWS layanan.
4. Di bawah Kasus penggunaan, pilih Lambda.

5. Pilih Berikutnya.
6. Pilih kebijakan berikut:
   - *clusterAuthPolicy*
   - AWSLambdaMSKExecutionRole
7. Pilih Berikutnya.
8. Untuk nama Peran, *lambdaAuthRole* masukkan lalu pilih Buat peran.

Untuk informasi selengkapnya, lihat the section called “Peran eksekusi (izin untuk fungsi untuk mengakses sumber daya lain)”.

Buat Fungsi Acara Lambda untuk membaca dari topik MSK Amazon Anda

Buat Fungsi Peristiwa Lambda yang dikonfigurasi untuk menggunakan peran IAM Anda. Anda dapat membuat fungsi Anda menggunakan konsol.

Untuk membuat fungsi menggunakan konfigurasi autentikasi Anda

1. Buka konsol Lambda dan pilih Create Event Function dari header.
2. Pilih Penulis dari awal.
3. Untuk nama Fungsi, berikan nama yang sesuai pilihan Anda.
4. Untuk Runtime, pilih versi terbaru yang didukung `Node.js` untuk menggunakan kode yang disediakan dalam tutorial ini.
5. Pilih Ubah peran eksekusi default.
6. Pilih Gunakan peran yang ada.
7. Untuk peran yang ada, pilih*lambdaAuthRole*.

Dalam lingkungan produksi, Anda biasanya perlu menambahkan kebijakan lebih lanjut ke peran eksekusi untuk Fungsi Peristiwa Lambda Anda untuk memproses peristiwa MSK Amazon Anda secara bermakna. Untuk informasi selengkapnya tentang menambahkan kebijakan ke peran Anda, lihat Menambahkan atau menghapus izin identitas dalam dokumentasi IAM.

Buat pemetaan sumber acara ke Fungsi Acara Lambda Anda

Pemetaan sumber peristiwa MSK Amazon Anda memberikan layanan Lambda informasi yang diperlukan untuk memanggil Lambda Anda saat peristiwa MSK Amazon yang sesuai terjadi. Anda

dapat membuat pemetaan MSK Amazon menggunakan konsol. Buat pemicu Lambda, lalu pemetaan sumber peristiwa diatur secara otomatis.

Untuk membuat pemicu Lambda (dan pemetaan sumber peristiwa)

1. Arahkan ke halaman ikhtisar Fungsi Acara Lambda Anda.
2. Di bagian Ikhtisar Fungsi Acara, pilih Tambahkan pemicu di kiri bawah.
3. Di menu tarik-turun Pilih sumber, pilih Amazon MSK.
4. Jangan mengatur otentikasi.
5. Untuk klaster MSK, pilih nama klaster Anda.
6. Untuk ukuran Batch, masukkan 1. Langkah ini membuat fitur ini lebih mudah untuk diuji, dan bukan merupakan nilai ideal dalam produksi.
7. Untuk nama Topik, berikan nama topik Kafka Anda.
8. Untuk ID grup Konsumen, berikan id grup konsumen Kafka Anda.

Perbarui Fungsi Acara Lambda Anda untuk membaca data streaming Anda

Lambda memberikan informasi tentang peristiwa Kafka melalui parameter metode acara. Untuk contoh struktur acara MSK Amazon, lihatthe section called “Contoh peristiwa”. Setelah Anda memahami cara menafsirkan peristiwa MSK Amazon yang diteruskan Lambda, Anda dapat mengubah kode Fungsi Peristiwa Lambda Anda untuk menggunakan informasi yang mereka berikan.

Berikan kode berikut ke Fungsi Peristiwa Lambda Anda untuk mencatat konten acara MSK Lambda Amazon untuk tujuan pengujian:

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara MSK Amazon dengan Lambda menggunakan.NET.

```
using System.Text;
```

```
using Amazon.Lambda.Core;
using Amazon.Lambda.KafkaEvents;


// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri

namespace MSKLambda;

public class Function
{


    /// <param name="input">The event for the Lambda function handler to
 process.</param>
    /// <param name="context">The ILambdaContext that provides methods for
 logging and describing the Lambda environment.</param>
    /// <returns></returns>
    public void FunctionHandler(KafkaEvent evnt, ILambdaContext context)
    {

        foreach (var record in evnt.Records)
        {
            Console.WriteLine("Key:" + record.Key);
            foreach (var eventRecord in record.Value)
            {
                var valueBytes = eventRecord.Value.ToArray();
                var valueText = Encoding.UTF8.GetString(valueBytes);

                Console.WriteLine("Message:" + valueText);
            }
        }
    }


}
```

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara MSK Amazon dengan Lambda menggunakan Go.

```
package main

import (
 "encoding/base64"
 "fmt"

 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

func handler(event events.KafkaEvent) {
 for key, records := range event.Records {
  fmt.Println("Key:", key)

  for _, record := range records {
   fmt.Println("Record:", record)

   decodedValue, _ := base64.StdEncoding.DecodeString(record.Value)
   message := string(decodedValue)
   fmt.Println("Message:", message)
  }
 }
}

func main() {
 lambda.Start(handler)
}
```

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara MSK Amazon dengan Lambda menggunakan Java.

```
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.KafkaEvent;
import com.amazonaws.services.lambda.runtime.events.KafkaEvent.KafkaEventRecord;

import java.util.Base64;
import java.util.Map;

public class Example implements RequestHandler<KafkaEvent, Void> {

    @Override
    public Void handleRequest(KafkaEvent event, Context context) {
        for (Map.Entry<String, java.util.List<KafkaEventRecord>> entry :
 event.getRecords().entrySet()) {
            String key = entry.getKey();
            System.out.println("Key: " + key);

            for (KafkaEventRecord record : entry.getValue()) {
                System.out.println("Record: " + record);

                byte[] value = Base64.getDecoder().decode(record.getValue());
                String message = new String(value);
                System.out.println("Message: " + message);
            }
        }

        return null;
    }
}
```

JavaScript

SDK untuk JavaScript (v3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara MSK Amazon dengan menggunakan JavaScript Lambda.

```
exports.handler = async (event) => {
    // Iterate through keys
    for (let key in event.records) {
      console.log('Key: ', key)
      // Iterate through records
      event.records[key].map((record) => {
        console.log('Record: ', record)
        // Decode base64
        const msg = Buffer.from(record.value, 'base64').toString()
        console.log('Message:', msg)
      })
    }
}
```

Mengkonsumsi acara MSK Amazon dengan menggunakan TypeScript Lambda.

```
import { MSKEvent, Context } from "aws-lambda";
import { Buffer } from "buffer";
import { Logger } from "@aws-lambda-powertools/logger";

const logger = new Logger({
  logLevel: "INFO",
  serviceName: "msk-handler-sample",
});

export const handler = async (
```

```
  event: MSKEvent,
  context: Context
): Promise<void> => {
  for (const [topic, topicRecords] of Object.entries(event.records)) {
    logger.info(`Processing key: ${topic}`);

    // Process each record in the partition
    for (const record of topicRecords) {
      try {
        // Decode the message value from base64
        const decodedMessage = Buffer.from(record.value, 'base64').toString();

        logger.info({
          message: decodedMessage
        });
      }
      catch (error) {
        logger.error('Error processing event', { error });
        throw error;
      }
    };
  }
}
```

PHP

SDK untuk PHP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara MSK Amazon dengan Lambda menggunakan PHP.

```
<?php
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

// using bref/bref and bref/logger for simplicity
```

```
use Bref\Context\Context;
use Bref\Event\Kafka\KafkaEvent;
use Bref\Event\Handler as StdHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler implements StdHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handle(mixed $event, Context $context): void
    {
        $kafkaEvent = new KafkaEvent($event);
        $this->logger->info("Processing records");
        $records = $kafkaEvent->getRecords();

        foreach ($records as $record) {
            try {
                $key = $record->getKey();
                $this->logger->info("Key: $key");

                $values = $record->getValue();
                $this->logger->info(json_encode($values));

                foreach ($values as $value) {
                    $this->logger->info("Value: $value");
                }

            } catch (Exception $e) {
                $this->logger->error($e->getMessage());
            }
        }
        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords records");
```

```
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara MSK Amazon dengan Lambda menggunakan Python.

```
import base64

def lambda_handler(event, context):
    # Iterate through keys
    for key in event['records']:
        print('Key:', key)
        # Iterate through records
        for record in event['records'][key]:
            print('Record:', record)
            # Decode base64
            msg = base64.b64decode(record['value']).decode('utf-8')
            print('Message:', msg)
```

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara MSK Amazon dengan Lambda menggunakan Ruby.

```
require 'base64'

def lambda_handler(event:, context:)
  # Iterate through keys
  event['records'].each do |key, records|
    puts "Key: #{key}"

    # Iterate through records
    records.each do |record|
      puts "Record: #{record}"

      # Decode base64
      msg = Base64.decode64(record['value'])
      puts "Message: #{msg}"
    end
  end
end
```

Rust

SDK for Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara MSK Amazon dengan Lambda menggunakan Rust.

```
use aws_lambda_events::event::kafka::KafkaEvent;
use lambda_runtime::{run, service_fn, tracing, Error, LambdaEvent};
use base64::prelude::*;
use serde_json::{Value};
use tracing::{info};

/// Pre-Requisites:
/// 1. Install Cargo Lambda - see https://www.cargo-lambda.info/guide/getting-
started.html
/// 2. Add packages tracing, tracing-subscriber, serde_json, base64
///
/// This is the main body for the function.
/// Write your code inside it.
/// There are some code example in the following URLs:
/// - https://github.com/awslabs/aws-lambda-rust-runtime/tree/main/examples
/// - https://github.com/aws-samples/serverless-rust-demo/

async fn function_handler(event: LambdaEvent<KafkaEvent>) -> Result<Value, Error>
 {

    let payload = event.payload.records;

    for (_name, records) in payload.iter() {

        for record in records {

         let record_text = record.value.as_ref().ok_or("Value is None")?;
         info!("Record: {}", &record_text);

         // perform Base64 decoding
         let record_bytes = BASE64_STANDARD.decode(record_text)?;
         let message = std::str::from_utf8(&record_bytes)?;

         info!("Message: {}", message);
        }

    }

    Ok(().into())
}

#[tokio::main]
```

```
async fn main() -> Result<(), Error> {

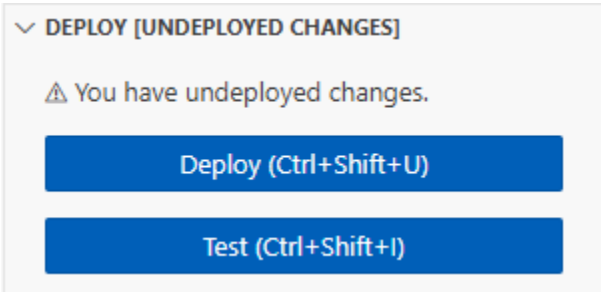
    // required to enable CloudWatch error logging by the runtime
    tracing::init_default_subscriber();
    info!("Setup CW subscriber!");

    run(service_fn(function_handler)).await
}
```

Anda dapat memberikan kode fungsi ke Lambda Anda menggunakan konsol.

Untuk memperbarui kode fungsi menggunakan editor kode konsol

1. Buka halaman Fungsi Acara konsol Lambda dan pilih fungsi Anda.
2. Pilih tab Kode.
3. Di panel Sumber kode, pilih file kode sumber Anda dan edit di editor kode terintegrasi.
4. Di bagian DEPLOY, pilih Deploy untuk memperbarui kode fungsi Anda:



Uji Fungsi Acara Lambda Anda untuk memverifikasi bahwa itu terhubung ke topik MSK Amazon Anda

Anda sekarang dapat memverifikasi apakah Lambda Anda sedang dipanggil oleh sumber peristiwa dengan memeriksa CloudWatch log peristiwa atau tidak.

Untuk memverifikasi apakah Fungsi Peristiwa Lambda Anda sedang dipanggil

1. Gunakan host admin Kafka Anda untuk menghasilkan acara Kafka menggunakan CLI`kafka-console-producer`. Untuk informasi lebih lanjut, lihat Menulis beberapa peristiwa ke dalam topik dalam dokumentasi Kafka. Kirim peristiwa yang cukup untuk mengisi batch yang ditentukan berdasarkan ukuran batch untuk pemetaan sumber peristiwa yang ditentukan pada langkah sebelumnya, atau Lambda menunggu informasi lebih lanjut untuk dipanggil.

2. Jika fungsi Anda berjalan, Lambda menulis apa yang terjadi. CloudWatch Di konsol, navigasikan ke halaman detail Fungsi Acara Lambda Anda.
3. Pilih tab Konfigurasi.
4. Dari sidebar, pilih Alat pemantauan dan operasi.
5. Identifikasi grup CloudWatch log di bawah konfigurasi Logging. Grup log harus dimulai dengan`/aws/lambda`. Pilih tautan ke grup log.
6. Di CloudWatch konsol, periksa peristiwa Log untuk peristiwa log yang dikirim Lambda ke aliran log. Identifikasi apakah ada peristiwa log yang berisi pesan dari acara Kafka Anda, seperti pada gambar berikut. Jika ada, Anda telah berhasil menghubungkan Fungsi Peristiwa Lambda ke Amazon MSK dengan pemetaan sumber peristiwa Lambda.



Menggunakan Lambda dengan Apache Kafka yang dikelola sendiri

Topik ini menjelaskan cara menggunakan Lambda dengan klaster Kafka yang dikelola sendiri. Dalam AWS terminologi, cluster yang dikelola sendiri mencakup cluster Kafka yang tidak AWS dihosting. Misalnya, Anda dapat meng-host cluster Kafka Anda dengan penyedia cloud seperti Confluent Cloud atau Redpanda.

Bab ini menjelaskan cara menggunakan cluster Apache Kafka yang dikelola sendiri sebagai sumber acara untuk Fungsi Acara Lambda Anda. Proses umum untuk mengintegrasikan Apache Kafka yang dikelola sendiri dengan Lambda melibatkan langkah-langkah berikut:

1. Pengaturan cluster dan jaringan — Pertama, siapkan cluster Apache Kafka yang dikelola sendiri dengan konfigurasi jaringan yang benar untuk memungkinkan Lambda mengakses cluster Anda.
2. Pengaturan pemetaan sumber peristiwa — Kemudian, buat sumber daya pemetaan sumber acara yang dibutuhkan Lambda untuk menghubungkan cluster Apache Kafka Anda dengan aman ke fungsi Anda.
3. Fungsi Acara dan pengaturan izin — Terakhir, pastikan bahwa fungsi Anda diatur dengan benar, dan memiliki izin yang diperlukan dalam peran pelaksanaannya.

Apache Kafka sebagai sumber peristiwa beroperasi sama dengan menggunakan Amazon Simple Queue Service (Amazon SQS) atau Amazon Kinesis. Lambda secara internal melakukan polling untuk pesan baru dari sumber acara dan kemudian secara sinkron memanggil Fungsi Peristiwa Lambda target. Lambda membaca pesan dalam batch dan menyediakan ini untuk fungsi Anda sebagai muatan acara. Ukuran batch maksimum dapat dikonfigurasi (defaultnya adalah 100 pesan). Untuk informasi selengkapnya, lihat the section called “Perilaku batching”.

Untuk mengoptimalkan throughput pemetaan sumber acara Apache Kafka yang dikelola sendiri, konfigurasikan mode yang disediakan. Dalam mode yang disediakan, Anda dapat menentukan jumlah minimum dan maksimum poller peristiwa yang dialokasikan untuk pemetaan sumber acara Anda. Ini dapat meningkatkan kemampuan pemetaan sumber acara Anda untuk menangani lonjakan pesan yang tidak terduga. Untuk informasi selengkapnya, lihat mode yang disediakan.

> **Warning**
>
> Pemetaan sumber peristiwa Lambda memproses setiap peristiwa setidaknya sekali, dan pemrosesan duplikat catatan dapat terjadi. Untuk menghindari potensi masalah yang terkait dengan duplikat peristiwa, kami sangat menyarankan agar Anda membuat kode fungsi Anda

idempoten. Untuk mempelajari lebih lanjut, lihat Bagaimana cara membuat fungsi Lambda saya idempoten di Pusat Pengetahuan. AWS

Untuk sumber Kafka-based peristiwa, Lambda mendukung parameter kontrol pemrosesan, seperti jendela batching dan ukuran batch. Untuk informasi selengkapnya, lihat Perilaku batching.

Untuk contoh cara menggunakan Kafka yang dikelola sendiri sebagai sumber acara, lihat Menggunakan Apache Kafka yang dihosting sendiri sebagai sumber acara di Blog Komputasi. AWS Lambda AWS

Topik

- Contoh peristiwa
- Mengkonfigurasi cluster dan jaringan Apache Kafka yang dikelola sendiri untuk Lambda
- Mengonfigurasi izin peran eksekusi Lambda
- Mengkonfigurasi sumber peristiwa Apache Kafka yang dikelola sendiri untuk Lambda

Contoh peristiwa

Lambda mengirimkan kumpulan pesan dalam parameter peristiwa saat memanggil Fungsi Peristiwa Lambda Anda. Muatan peristiwa berisi array pesan. Setiap item array berisi detail dari topik Kafka dan pengidentifikasi Kafka partisi, bersama-sama dengan stempel waktu dan pesan berkode base64.

```
{
   "eventSource": "SelfManagedKafka",
   "bootstrapServers":"b-2.demo-cluster-1.a1bcde.c1.kafka.us-
east-1.amazonaws.com:9092,b-1.demo-cluster-1.a1bcde.c1.kafka.us-
east-1.amazonaws.com:9092",
   "records":{
      "mytopic-0":[
         {
            "topic":"mytopic",
            "partition":0,
            "offset":15,
            "timestamp":1545084650987,
            "timestampType":"CREATE_TIME",
            "key":"abcDEFghiJKLmnoPQRstuVWXyz1234==",
            "value":"SGVsbG8sIHRoaXMgaXMgYSB0ZXN0Lg==",
            "headers":[
```

```
            {
               "headerKey":[
                  104,
                  101,
                  97,
                  100,
                  101,
                  114,
                  86,
                  97,
                  108,
                  117,
                  101
               ]
            }
         ]
      }
   ]
}
}
```

Mengkonfigurasi cluster dan jaringan Apache Kafka yang dikelola sendiri untuk Lambda

Untuk menghubungkan Fungsi Acara Lambda Anda ke cluster Apache Kafka yang dikelola sendiri, Anda perlu mengkonfigurasi cluster dan jaringan tempat ia berada dengan benar. Halaman ini menjelaskan cara mengkonfigurasi cluster dan jaringan Anda. Jika cluster dan jaringan Anda sudah dikonfigurasi dengan benar, lihat the section called “Konfigurasikan sumber acara” untuk mengonfigurasi pemetaan sumber peristiwa.

Topik

- Self-managed Pengaturan kluster Apache Kafka
- Konfigurasikan keamanan jaringan

Self-managed Pengaturan kluster Apache Kafka

Anda dapat meng-host cluster Apache Kafka yang dikelola sendiri dengan penyedia cloud seperti Confluent Cloud atau Redpanda, atau menjalankannya di infrastruktur Anda sendiri. Pastikan klaster Anda dikonfigurasi dengan benar dan dapat diakses dari jaringan tempat pemetaan sumber peristiwa Lambda Anda terhubung.

Konfigurasikan keamanan jaringan

Untuk memberi Lambda akses penuh ke Apache Kafka yang dikelola sendiri melalui pemetaan sumber acara Anda, kluster Anda harus menggunakan titik akhir publik (alamat IP publik), atau Anda harus memberikan akses ke Amazon VPC tempat Anda membuat cluster.

Saat Anda menggunakan Apache Kafka yang dikelola sendiri dengan Lambda, buat titik akhir AWS PrivateLink VPC yang menyediakan akses Fungsi Acara ke sumber daya di VPC Amazon Anda.

> **Note**
>
> AWS PrivateLink Titik akhir VPC diperlukan untuk fungsi dengan pemetaan sumber peristiwa yang menggunakan mode default (sesuai permintaan) untuk poller acara. Jika pemetaan sumber acara menggunakan mode yang disediakan, Anda tidak perlu mengonfigurasi titik akhir AWS PrivateLink VPC.

Buat titik akhir untuk menyediakan akses ke sumber daya berikut:

- Lambda — Buat titik akhir untuk kepala layanan Lambda.
- AWS STS — Buat titik akhir AWS STS untuk prinsipal layanan untuk mengambil peran atas nama Anda.
- Secrets Manager - Jika klaster Anda menggunakan Secrets Manager untuk menyimpan kredensi, buat endpoint untuk Secrets Manager.

Atau, konfigurasikan gateway NAT di setiap subnet publik di Amazon VPC. Untuk informasi selengkapnya, lihat the section called “Akses internet untuk fungsi VPC”.

Saat Anda membuat pemetaan sumber peristiwa untuk Apache Kafka yang dikelola sendiri, Lambda memeriksa apakah Antarmuka Jaringan Elastis (ENI) sudah ada untuk subnet dan grup keamanan yang dikonfigurasi untuk VPC Amazon Anda. Jika Lambda menemukan ENI yang ada, ia mencoba untuk menggunakannya kembali. Jika tidak, Lambda membuat ENI baru untuk terhubung ke sumber acara dan menjalankan fungsi Anda.

> **Note**
>
> Fungsi Acara Lambda selalu berjalan di dalam VPC yang dimiliki oleh layanan Lambda. Konfigurasi VPC fungsi Anda tidak memengaruhi pemetaan sumber peristiwa. Hanya

konfigurasi jaringan sumber acara yang menentukan bagaimana Lambda terhubung ke sumber acara Anda.

Konfigurasikan grup keamanan untuk VPC Amazon yang berisi cluster Anda. Secara default, Apache Kafka yang dikelola sendiri menggunakan port berikut:. `9092`

- Aturan masuk - Izinkan semua lalu lintas di port broker default untuk grup keamanan yang terkait dengan sumber acara Anda. Atau, Anda dapat menggunakan aturan grup keamanan referensi mandiri untuk mengizinkan akses dari instans dalam grup keamanan yang sama.
- Aturan keluar — Izinkan semua lalu lintas di port `443` untuk tujuan eksternal jika fungsi Anda perlu berkomunikasi dengan AWS layanan. Atau, Anda juga dapat menggunakan aturan grup keamanan referensi diri untuk membatasi akses ke broker jika Anda tidak perlu berkomunikasi dengan layanan lain AWS .
- Aturan masuk titik akhir VPC Amazon — Jika Anda menggunakan titik akhir VPC Amazon, grup keamanan yang terkait dengan titik akhir VPC Amazon Anda harus mengizinkan lalu lintas masuk pada port dari grup keamanan klaster. `443`

Jika klaster menggunakan autentikasi, Anda juga dapat membatasi kebijakan endpoint untuk titik akhir Secrets Manager. Untuk memanggil Secrets Manager API, Lambda menggunakan peran fungsi Anda, bukan kepala layanan Lambda.

Example Kebijakan titik akhir VPC — titik akhir Secrets Manager

```
{
      "Statement": [
          {
              "Action": "secretsmanager:GetSecretValue",
              "Effect": "Allow",
              "Principal": {
                  "AWS": [
                      "arn:aws::iam::123456789012:role/my-role"
                  ]
              },
              "Resource": "arn:aws::secretsmanager:us-west-2:123456789012:secret:my-
secret"
          }
      ]
  }
```

Saat Anda menggunakan titik akhir Amazon VPC, AWS merutekan panggilan API Anda untuk menjalankan fungsi Anda menggunakan Antarmuka Jaringan Elastis (ENI) titik akhir. Kepala layanan Lambda perlu memanggil `lambda:InvokeFunction` peran dan fungsi apa pun yang menggunakan ENI tersebut.

Secara default, titik akhir Amazon VPC memiliki kebijakan IAM terbuka yang memungkinkan akses luas ke sumber daya. Praktik terbaik adalah membatasi kebijakan ini untuk melakukan tindakan yang diperlukan menggunakan titik akhir tersebut. Untuk memastikan bahwa pemetaan sumber peristiwa Anda dapat menjalankan Fungsi Peristiwa Lambda Anda, kebijakan titik akhir VPC harus mengizinkan kepala layanan Lambda untuk memanggil dan. `sts:AssumeRole` `lambda:InvokeFunction` Membatasi kebijakan titik akhir VPC Anda agar hanya mengizinkan panggilan API yang berasal dari organisasi Anda mencegah pemetaan sumber peristiwa berfungsi dengan baik, `"Resource": "*"` sehingga diperlukan dalam kebijakan ini.

Contoh kebijakan titik akhir VPC berikut menunjukkan cara memberikan akses yang diperlukan ke prinsipal layanan Lambda untuk titik akhir dan Lambda. AWS STS

Example Kebijakan Titik Akhir VPC — AWS STS titik akhir

```
{
      "Statement": [
          {
              "Action": "sts:AssumeRole",
              "Effect": "Allow",
              "Principal": {
                  "Service": [
                      "lambda.amazonaws.com"
                  ]
              },
              "Resource": "*"
          }
      ]
    }
```

Example Kebijakan Titik Akhir VPC - Titik akhir Lambda

```
{
      "Statement": [
          {
              "Action": "lambda:InvokeFunction",
              "Effect": "Allow",
```

```
            "Principal": {
                "Service": [
                    "lambda.amazonaws.com"
                ]
            },
            "Resource": "*"
        }
    ]
}
```

Mengonfigurasi izin peran eksekusi Lambda

Selain mengakses cluster Kafka yang dikelola sendiri, Fungsi Acara Lambda Anda memerlukan izin untuk melakukan berbagai tindakan API. Anda menambahkan izin ini ke peran eksekusi fungsi. Jika pengguna Anda memerlukan akses ke tindakan API apa pun, tambahkan izin yang diperlukan ke kebijakan identitas untuk pengguna atau AWS Identity and Access Management peran (IAM).

Topik

- Izin Fungsi Acara Lambda yang diperlukan
- Izin Fungsi Acara Lambda opsional
- Menambahkan izin ke peran eksekusi
- Memberikan akses pengguna dengan kebijakan IAM

Izin Fungsi Acara Lambda yang diperlukan

Untuk membuat dan menyimpan log dalam grup log di Amazon CloudWatch Logs, Fungsi Peristiwa Lambda Anda harus memiliki izin berikut dalam peran pelaksanaannya:

- log: CreateLogGroup
- log: CreateLogStream
- log: PutLogEvents

Izin Fungsi Acara Lambda opsional

Fungsi Acara Lambda Anda mungkin juga memerlukan izin untuk:

- Jelaskan rahasia Secrets Manager Anda.
- Akses AWS Key Management Service (AWS KMS) kunci terkelola pelanggan Anda.

- Akses VPC Amazon Anda.
- Kirim catatan pemanggilan yang gagal ke tujuan.

Secrets Manager dan AWS KMS izin

Bergantung pada jenis kontrol akses yang Anda konfigurasikan untuk broker Kafka Anda, Fungsi Acara Lambda Anda mungkin memerlukan izin untuk mengakses rahasia Secrets Manager Anda atau untuk mendekripsi kunci yang dikelola pelanggan Anda. AWS KMS Untuk mengakses sumber daya ini, peran eksekusi fungsi Anda harus memiliki izin berikut:

- manajer rahasia: GetSecretValue
- kms:Decrypt

Izin VPC

Jika hanya pengguna dalam VPC yang dapat mengakses cluster Apache Kafka yang dikelola sendiri, Fungsi Acara Lambda Anda harus memiliki izin untuk mengakses sumber daya VPC Amazon Anda. Sumber daya ini termasuk VPC, subnet, grup keamanan, dan antarmuka jaringan. Untuk mengakses sumber daya ini, peran eksekusi fungsi Anda harus memiliki izin berikut:

- EC2: CreateNetworkInterface
- EC2: DescribeNetworkInterfaces
- EC2: DescribeVpcs
- EC2: DeleteNetworkInterface
- EC2: DescribeSubnets
- EC2: DescribeSecurityGroups

Menambahkan izin ke peran eksekusi

Untuk mengakses AWS layanan lain yang digunakan cluster Apache Kafka yang dikelola sendiri, Lambda menggunakan kebijakan izin yang Anda tetapkan dalam peran eksekusi Fungsi Peristiwa Lambda.

Secara default, Lambda tidak diizinkan untuk melakukan tindakan yang diperlukan atau opsional untuk klaster Apache Kafka yang dikelola sendiri. Anda harus membuat dan mendefinisikan tindakan ini dalam kebijakan kepercayaan IAM untuk peran eksekusi Anda. Contoh ini menunjukkan cara agar

Anda dapat membuat kebijakan yang mengizinkan Lambda mengakses sumber daya Amazon VPC Anda.

JSON

```
{
        "Version":"2012-10-17",
        "Statement":[
           {
              "Effect":"Allow",
              "Action":[
                 "ec2:CreateNetworkInterface",
                 "ec2:DescribeNetworkInterfaces",
                 "ec2:DescribeVpcs",
                 "ec2:DeleteNetworkInterface",
                 "ec2:DescribeSubnets",
                 "ec2:DescribeSecurityGroups"
              ],
              "Resource":"*"
           }
        ]
     }
```

Memberikan akses pengguna dengan kebijakan IAM

Secara default, pengguna dan peran tidak memiliki izin untuk melakukan operasi API sumber peristiwa. Untuk memberikan akses ke pengguna di organisasi atau akun Anda, Anda membuat atau memperbarui kebijakan berbasis identitas. Untuk informasi selengkapnya, lihat Mengontrol akses ke AWS sumber daya menggunakan kebijakan di Panduan Pengguna IAM.

Untuk mengatasi masalah kesalahan autentikasi dan otorisasi, lihat. the section called “Pemecahan masalah”

Mengkonfigurasi sumber peristiwa Apache Kafka yang dikelola sendiri untuk Lambda

Untuk menggunakan cluster Apache Kafka yang dikelola sendiri sebagai sumber peristiwa untuk Lambda Event Function Anda, Anda membuat pemetaan sumber acara yang menghubungkan kedua sumber daya tersebut. Halaman ini menjelaskan cara membuat pemetaan sumber peristiwa untuk Apache Kafka yang dikelola sendiri.

Halaman ini mengasumsikan bahwa Anda telah mengkonfigurasi klaster Kafka Anda dengan benar dan jaringan yang berada di dalamnya. Jika Anda perlu mengatur cluster atau jaringan Anda, lihat the section called “Pengaturan cluster dan jaringan”.

Topik

- Menggunakan cluster Apache Kafka yang dikelola sendiri sebagai sumber peristiwa
- Mengkonfigurasi metode otentikasi cluster di Lambda
- Membuat pemetaan sumber peristiwa Lambda untuk sumber peristiwa Apache Kafka yang dikelola sendiri
- Semua parameter konfigurasi sumber peristiwa Apache Kafka yang dikelola sendiri di Lambda

Menggunakan cluster Apache Kafka yang dikelola sendiri sebagai sumber peristiwa

Ketika Anda menambahkan cluster Apache Kafka atau Amazon MSK sebagai pemicu untuk Lambda Event Function Anda, cluster digunakan sebagai sumber acara.

Lambda membaca data peristiwa dari topik Kafka yang Anda tentukan sebagai `Topics` dalam CreateEventSourceMapping permintaan, berdasarkan posisi awal yang Anda tentukan. Setelah pemrosesan berhasil, topik Kafka Anda dijalankan untuk klaster Kafka Anda.

Lambda membaca pesan secara berurutan untuk setiap partisi topik Kafka. Satu muatan Lambda dapat berisi pesan dari beberapa partisi. Ketika lebih banyak catatan tersedia, Lambda terus memproses catatan dalam batch, berdasarkan BatchSize nilai yang Anda tentukan dalam CreateEventSourceMapping permintaan, sampai fungsi Anda mengejar topik.

Setelah Lambda memproses setiap batch, Lambda melakukan offset pesan dalam batch tersebut. Jika fungsi Anda mengembalikan kesalahan untuk salah satu pesan dalam batch, Lambda mencoba ulang seluruh batch pesan sampai berhasil diproses atau pesan berakhir. Anda dapat mengirim catatan yang gagal dalam semua upaya percobaan ulang ke tujuan saat gagal untuk diproses nanti.

> **Note**
>
> Sementara Lambda Event Functions biasanya memiliki batas waktu tunggu maksimum 15 menit, pemetaan sumber peristiwa untuk Amazon MSK, Apache Kafka yang dikelola sendiri, Amazon DocumentDB, dan Amazon MQ untuk ActiveMQ dan RabbitMQ hanya mendukung fungsi dengan batas waktu tunggu maksimum 14 menit.

Mengkonfigurasi metode otentikasi cluster di Lambda

Lambda mendukung beberapa metode untuk mengautentikasi dengan klaster Apache Kafka yang dikelola sendiri. Pastikan Anda mengonfigurasi klaster Kafka untuk menggunakan salah satu metode otentikasi yang didukung ini. Untuk informasi lebih lanjut tentang keamanan Kafka, lihat Keamanan bagian dari dokumentasi Kafka.

SASL/SCRAM otentikasi

Lambda mendukung Otentikasi dan Keamanan Sederhana Layer/Salted Mekanisme Otentikasi Respon Tantangan (SASL/SCRAM) otentikasi dengan enkripsi Transport Layer Security (TLS) (`SASL_SSL`). Lambda mengirimkan kredensial terenkripsi untuk mengautentikasi dengan klaster. Lambda tidak mendukung SASL/SCRAM dengan plaintext (`SASL_PLAINTEXT`). Untuk informasi lebih lanjut tentang SASL/SCRAM otentikasi, lihat RFC 5802.

Lambda juga mendukung SASL/PLAIN otentikasi. Karena mekanisme ini menggunakan kredentif teks jelas, koneksi ke server harus menggunakan enkripsi TLS untuk memastikan bahwa kredenSIAL dilindungi.

Untuk otentikasi SASL, Anda menyimpan kredentif masuk sebagai rahasia di AWS Secrets Manager. Untuk informasi selengkapnya tentang menggunakan Secrets Manager, lihat Buat sebuah AWS Secrets Manager rahasia di AWS Secrets Manager Panduan Pengguna.

> **Important**
>
> Untuk menggunakan Secrets Manager untuk otentikasi, rahasia harus disimpan di tempat yang sama AWS wilayah sebagai Fungsi Peristiwa Lambda Anda.

Autentikasi TLS bersama

Mutual TLS (MTLS) menyediakan otentikasi dua arah antara klien dan server. Klien mengirim sertifikat ke server untuk server untuk memverifikasi klien, dan server mengirimkan sertifikat ke klien untuk klien untuk memverifikasi server.

Dalam Apache Kafka yang dikelola sendiri, Lambda bertindak sebagai klien. Anda mengonfigurasi sertifikat klien (sebagai rahasia di Secrets Manager) untuk mengotentikasi Lambda dengan broker Kafka Anda. Sertifikat klien harus ditandatangani oleh CA di penyimpanan kepercayaan server.

Cluster Kafka mengirimkan sertifikat server ke Lambda untuk mengotentikasi broker Kafka dengan Lambda. Sertifikat server dapat berupa sertifikat CA publik atau pribadi CA/self-signed sertifikat.

Sertifikat CA publik harus ditandatangani oleh otoritas sertifikat (CA) yang ada di penyimpanan kepercayaan Lambda. Untuk pribadi CA/self-signed sertifikat, Anda mengkonfigurasi sertifikat CA root server (sebagai rahasia di Secrets Manager). Lambda menggunakan sertifikat root untuk memverifikasi broker Kafka.

Untuk informasi lebih lanjut tentang MTL, lihat Memperkenalkan otentikasi TLS bersama untuk Amazon MSK sebagai sumber acara.

Autentikasi OAuth 2.0

Dukungan Lambda SASL/OAUTHBEARER otentikasi dengan enkripsi TLS (`SASL_SSL`). Lambda meminta token akses dari penyedia identitas OAuth 2.0 Anda dan menyajikan token itu kepada broker Kafka Anda. Lambda menyegarkan token sebelum kedaluwarsa. Konfigurasikan broker Anda untuk memvalidasi token dari penyedia identitas yang sama.

Untuk menggunakan metode ini, pilih `OAUTHBEARER_AUTH` sebagai `Type` dari a SourceAccessConfiguration dan berikan Arn Manajer Rahasia dari rahasia OAuth Anda di `URI` bidang. Untuk isi rahasianya, lihat the section called “Mengkonfigurasi rahasia OAuth”.

Lambda mendukung dua jenis hibah. Bidang yang Anda simpan di rahasia menentukan mana yang digunakan Lambda.

- Kredensibilitas klien — Lambda mengirimkan ID klien dan rahasia klien ke titik akhir token. Gunakan jenis hibah ini saat penyedia identitas Anda mengeluarkan rahasia klien.
- Pembawa JWT — Lambda menandatangani pernyataan JSON Web Token (JWT) dengan kunci pribadi dan mengirimkan pernyataan ke titik akhir token. Gunakan jenis hibah ini ketika penyedia identitas Anda mengharapkan pernyataan yang ditandatangani, bukan rahasia klien.

Jika penyedia identitas Anda atau broker Anda memerlukan nilai tambahan, berikan masing-masing sebagai entri konfigurasi akses sumbernya sendiri. Masukkan nilai literal di `URI` lapangan bukan ARN rahasia.

- `OAUTHBEARER_SCOPE` — Ruang lingkup yang diminta Lambda ketika meminta token.
- `OAUTHBEARER_AUDIENCE` — Audiens yang diminta Lambda ketika meminta token.
- `OAUTHBEARER_LOGICAL_CLUSTER` — ID cluster logis yang dikirim Lambda ke broker Anda sebagai ekstensi SASL.
- `OAUTHBEARER_IDENTITY_POOL` — ID kumpulan identitas yang dikirim Lambda ke broker Anda sebagai ekstensi SASL.

> **Note**
>
> Keempat jenis ini memerlukan jenis otentikasi OAuth 2.0. Berikan mereka `OAUTHBEARER_AUTH`, atau hanya menyediakan `OAUTHBEARER_AUDIENCE` bersama `IAM_OAUTHBEARER_AUTH`. Jika Anda menyediakan salah satu dari mereka tanpa salah satu jenis otentikasi ini, Lambda mengembalikan kesalahan validasi.

Jika broker Anda menunjukkan sertifikat yang ditandatangani oleh CA swasta, berikan juga `SERVER_ROOT_CA_CERTIFICATE` sehingga Lambda dapat memverifikasinya. Untuk informasi selengkapnya, lihat the section called “Mengkonfigurasi rahasia sertifikat CA root server”.

Autentikasi IAM

Jika cluster Anda adalah cluster Amazon MSK yang Anda jangkau melalui pengaturan jaringan Anda sendiri, Anda dapat mendaftarkannya sebagai sumber peristiwa Kafka yang dikelola sendiri dan mengotentikasi dengan IAM. Pengaturan jaringan khusus mencakup nama domain khusus di depan broker Anda atau koneksi dari akun lain. Lambda menandatangani setiap koneksi dengan kredensional peran eksekusi fungsi Anda, sehingga Anda tidak menyimpan kredenSIAL apa pun di Manajer Rahasia.

Untuk menggunakan metode ini, aktifkan kontrol akses IAM di cluster Anda. Kemudian pilih `IAM_AUTH` sebagai `Type` dari konfigurasi akses sumber dan hilangkan `URI` bidang. Peran eksekusi fungsi Anda memerlukan izin cluster yang dijelaskan di the section called “Mengatur Konfigurasi Izin ”.

Jika broker Anda menunjukkan sertifikat yang ditandatangani oleh CA swasta, berikan juga `SERVER_ROOT_CA_CERTIFICATE` sehingga Lambda dapat memverifikasinya. Nama domain khusus di depan broker Anda biasanya membutuhkan ini.

Otentikasi IAM dengan SASL/OAUTHBEARER

Jika broker Anda menerima token OAuth 2.0, Anda dapat menggunakan AWS sebagai penyedia identitas. Lambda menyajikan AWS token identitas web untuk broker Anda SASL/OAUTHBEARER. Lambda meminta token berumur pendek untuk peran eksekusi fungsi Anda dengan `sts:GetWebIdentityToken` tindakan, sehingga Anda tidak menyimpan kredenSIAL apa pun di Manajer Rahasia. Lambda meminta token baru sebelum token saat ini kedaluwarsa.

Untuk menggunakan metode ini, pilih `IAM_OAUTHBEARER_AUTH` sebagai `Type` dari konfigurasi akses sumber dan hilangkan `URI` bidang. Anda juga harus menyediakan `OAUTHBEARER_AUDIENCE`

dengan audiens yang diharapkan broker Anda. Peran eksekusi fungsi Anda memerlukan izin untuk memanggil `sts:GetWebIdentityToken`.

Konfigurasikan broker Anda untuk dipercaya AWS sebagai penyedia identitas OpenID Connect (OIDC) sebelum Anda membuat pemetaan sumber peristiwa. Jika broker Anda tidak mempercayai AWS penerbit, mereka menolak token dan Lambda tidak dapat membaca dari topik Anda.

> **Note**
>
> `IAM_OAUTHBEARER_AUTH` hanya mendukung `OAUTHBEARER_AUDIENCE`. Jika Anda juga menyediakan `OAUTHBEARER_SCOPE`, `OAUTHBEARER_LOGICAL_CLUSTER`, atau `OAUTHBEARER_IDENTITY_POOL`, Lambda mengembalikan kesalahan validasi.

Mengkonfigurasi rahasia sertifikat klien

Rahasia CLIENT_CERTIFICATE_TLS_AUTH memerlukan bidang sertifikat dan bidang kunci pribadi. Untuk kunci pribadi terenkripsi, rahasia memerlukan kata sandi kunci pribadi. Sertifikat dan kunci pribadi harus dalam format PEM.

> **Note**
>
> Lambda mendukung PBES1 (tetapi bukan PBES2) algoritma enkripsi kunci pribadi.

Bidang sertifikat harus berisi daftar sertifikat, dimulai dengan sertifikat klien, diikuti oleh sertifikat perantara, dan diakhiri dengan sertifikat root. Setiap sertifikat harus dimulai pada baris baru dengan struktur berikut:

```
-----BEGIN CERTIFICATE-----
            <certificate contents>
-----END CERTIFICATE-----
```

Secrets Manager mendukung rahasia hingga 65.536 byte, yang merupakan ruang yang cukup untuk rantai sertifikat yang panjang.

Kunci pribadi harus masuk PKCS #8 format, dengan struktur sebagai berikut:

```
-----BEGIN PRIVATE KEY-----
             <private key contents>
```

```
-----END PRIVATE KEY-----
```

Untuk kunci pribadi terenkripsi, gunakan struktur berikut:

```
-----BEGIN ENCRYPTED PRIVATE KEY-----
              <private key contents>
-----END ENCRYPTED PRIVATE KEY-----
```

Contoh berikut menunjukkan isi rahasia untuk otentikasi MTL menggunakan kunci pribadi terenkripsi. Untuk kunci pribadi terenkripsi, sertakan kata sandi kunci pribadi di rahasia.

```
{"privateKeyPassword":"testpassword",
"certificate":"-----BEGIN CERTIFICATE-----
MIIE5DCCAsygAwIBAgIRAPJdwaFaNRrytHBto0j5BA0wDQYJKoZIhvcNAQELBQAw
...
j0Lh4/+1HfgyE2KlmII36dg4IMzNjAFEBZiCRoPimO40s1cRqtFHXoal0QQbIlxk
cmUuiAii9R0=
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
MIIFgjCCA2qgAwIBAgIQdjNZd6uFf9hbNC5RdfmHrzANBgkqhkiG9w0BAQsFADBb
...
rQoiowbbk5wXCheYSANQIfTZ6weQTgiCHCCbuuMKNVS95FkXm0vqVD/YpXKwA/no
c8PH3PSoAaRwMMgOSA2ALJvbRz8mpg==
-----END CERTIFICATE-----",
"privateKey":"-----BEGIN ENCRYPTED PRIVATE KEY-----
MIIFKzBVBgkqhkiG9w0BBQ0wSDAnBgkqhkiG9w0BBQwwGgQUiAFcK5hT/X7Kjmgp
...
QrSekqF+kWzmB6nAfSzgO9IaoAaytLvNgGTckWeUkWn/V0Ck+LdGUXzAC4RxZnoQ
zp2mwJn2NYB7AZ7+imp0azDZb+8YG2aUCiyqb6PnnA==
-----END ENCRYPTED PRIVATE KEY-----"
}
```

Mengkonfigurasi rahasia sertifikat CA root server

Anda membuat rahasia ini jika broker Kafka Anda menggunakan enkripsi TLS dengan sertifikat yang ditandatangani oleh CA pribadi. Anda dapat menggunakan enkripsi TLS untuk VPC, SASL/SCRAM, SASL/PLAIN, atau otentikasi MTL.

Rahasia sertifikat CA root server memerlukan bidang yang berisi sertifikat CA root broker Kafka dalam format PEM. Contoh berikut menunjukkan struktur rahasia.

```
{"certificate":"-----BEGIN CERTIFICATE-----
```

```
MIID7zCCAtegAwIBAgIBADANBgkqhkiG9w0BAQsFADCBmDELMAkGA1UEBhMCVVMx
EDAOBgNVBAgTB0FyaXpvbmExEzARBgNVBAcTClNjb3R0c2RhbGUxJTAjBgNVBAoT
HFN0YXJmaWVsZCBUZWNobm9sb2dpZXMsIEluYy4xOzA5BgNVBAMTMlN0YXJmaWVs
ZCBTZXJ2aWNlcyBSb290IENlcnRpZmljYXRlIEF1dG...
-----END CERTIFICATE-----"
}
```

Mengkonfigurasi rahasia OAuth

Anda membuat rahasia ini saat Anda menggunakan `0AUTHBEARER_AUTH`. Rahasianya adalah dokumen JSON. Setiap kebutuhan rahasia OAuth `oauthTokenEndpointUrl` dan `oauthClientId`. Tambahkan bidang hanya untuk satu jenis hibah. Jika Anda memberikan bidang dari kedua jenis hibah, Lambda mengembalikan kesalahan validasi.

Bidang	Diperlukan	Deskripsi
`oauthTokenEndpoint Url`	Y	Titik akhir token penyedia identitas Anda. Ini harus berupa URL HTTPS.
`oauthClientId`	Y	ID klien yang dikeluarkan penyedia identitas Anda.
`oauthClientSecret`	Hanya kredensi klien	Rahasia klien yang dikeluark an penyedia identitas Anda.
`oauthClientEmail`	Hanya pembawa JWT	Identitas akun layanan yang dimasukkan Lambda dalam pernyataan JWT.
`oauthPrivateKey`	Hanya pembawa JWT	Kunci pribadi yang digunakan Lambda untuk menandata ngani pernyataan JWT, dalam format PEM.
`oauthTokenExpirati onSeconds`	T	Masa hidup pernyataan JWT yang ditandatangani Lambda. Hanya berlaku untuk jenis hibah pembawa JWT.

Bidang	Diperlukan	Deskripsi
`oauthIdpCaCertificate`	T	Sertifikat CA root dari penyedia identitas Anda, dalam format PEM. Berikan ini jika penyedia identitas Anda menunjukkan sertifikat yang ditandatangani oleh CA pribadi.

Contoh berikut menunjukkan rahasia untuk jenis pemberian kredenSIAL klien.

```
{"oauthTokenEndpointUrl":"https://idp.example.com/oauth2/token",
"oauthClientId":"my-client-id",
"oauthClientSecret":"my-client-secret"}
```

Contoh berikut menunjukkan rahasia untuk jenis hibah pembawa JWT.

```
{"oauthTokenEndpointUrl":"https://idp.example.com/oauth2/token",
"oauthClientId":"my-client-id",
"oauthClientEmail":"my-service-account@example.com",
"oauthPrivateKey":"-----BEGIN PRIVATE KEY-----
<private key contents>
-----END PRIVATE KEY-----"}
```

Important

Simpan rahasia di tempat yang sama AWS Wilayah sebagai fungsi Lambda Anda. Lambda membaca rahasia dengan peran eksekusi fungsi Anda, sehingga peran itu perlu `secretsmanager:GetSecretValue` Izin untuk Rahasianya

Anda dapat memutar kredenSIAL dalam rahasia tanpa membuat ulang pemetaan sumber peristiwa. Lambda mengambil nilai-nilai baru dan meminta token baru.

Membuat pemetaan sumber peristiwa Lambda untuk sumber peristiwa Apache Kafka yang dikelola sendiri

Untuk membuat pemetaan sumber acara, Anda dapat menggunakan konsol Lambda, AWS Command Line Interface (CLI), atau AWS SDK.

Langkah-langkah konsol berikut menambahkan cluster Apache Kafka yang dikelola sendiri sebagai pemicu untuk Lambda Event Function Anda. Di bawah kap, ini menciptakan sumber pemetaan sumber peristiwa.

Prasyarat

- Klaster Apache Kafka yang dikelola sendiri. Lambda mendukung Apache Kafka versi 0.10.1.0 dan yang lebih baru.
- Sebuah peran eksekusi dengan izin untuk mengakses AWS sumber daya yang digunakan cluster Kafka yang dikelola sendiri.

Menambahkan klaster Kafka yang dikelola sendiri (konsol)

Ikuti langkah-langkah ini untuk menambahkan cluster Apache Kafka yang dikelola sendiri dan topik Kafka sebagai pemicu untuk Lambda Event Function Anda.

Untuk menambahkan pemicu Apache Kafka ke Lambda Event Function (konsol) Anda

1. Buka Halaman Fungsi Acara dari konsol Lambda.
2. Pilih nama Lambda Event Function Anda.
3. Di bagian Gambaran umum fungsi, pilih Tambah pemicu.
4. Pada Konfigurasi pemicu, lakukan hal berikut:

    a. Pilih jenis pemicu Apache Kafka.
    b. Untuk Server bootstrap, masukkan alamat pasangan host dan port broker Kafka di klaster Anda, lalu pilih Tambahkan. Ulangi untuk setiap broker Kafka di klaster.
    c. Untuk Nama topik, masukkan nama topik Kafka yang digunakan untuk menyimpan catatan dalam klaster.
    d. Jika Anda mengonfigurasi mode yang disediakan, masukkan nilai untuk Poller acara minimum, nilai untuk Poller acara maksimum, dan nilai opsional untuk PollerGroupName untuk menentukan pengelompokan beberapa ESM dalam VPC sumber peristiwa yang sama.

e. (Opsional) Untuk Ukuran batch, masukkan jumlah maksimum catatan yang akan diterima dalam satu batch.

f. Untuk Jendela batch, masukkan jumlah detik maksimum yang dihabiskan Lambda untuk mengumpulkan catatan sebelum memanggil fungsi.

g. (Opsional) Untuk ID grup konsumen, masukkan ID grup konsumen Kafka untuk bergabung.

h. (Opsional) Untuk Posisi awal, pilih Terbaru untuk mulai membaca streaming dari catatan terbaru, Potong cakrawala untuk memulai pada catatan paling awal yang tersedia, atau Pada stempel waktu untuk menentukan stempel waktu untuk mulai membaca.

i. (Opsional) Untuk VPC, pilih Amazon VPC untuk cluster Kafka Anda. Kemudian, pilih Subnet VPC dan Grup keamanan VPC.

   Pengaturan ini diperlukan jika hanya pengguna dalam VPC Anda yang mengakses broker Anda.

j. (Opsional) Untuk Otentikasi, pilih Tambah, dan kemudian lakukan hal berikut:

   i. Pilih protokol akses atau otentikasi broker Kafka di cluster Anda.

      - Jika broker Kafka Anda menggunakan SASL/PLAIN otentikasi, pilih AUTENTIKASI DASAR.
      - Jika broker Anda menggunakan SASL/SCRAM otentikasi, pilih salah satu SASL_SCRAM protokol.
      - Jika Anda mengonfigurasi otentikasi MTL, pilih CLIENT_CERTIFICATE_TLS_AUTH protokol.
      - Jika broker Anda menggunakan OAuth 2.0, pilih OAUTHBEARER_AUTH.
      - Jika broker Anda menggunakan kontrol akses IAM, pilih IAM_AUTH.
      - Jika broker Anda menerima AWS token identitas web di atas OAuth 2.0, pilih IAM_OAUTHBEARER_AUTH.

   ii. Untuk SASL/SCRAM, MtLS, atau otentikasi OAuth 2.0, pilih kunci rahasia Manajer Rahasia yang berisi kredenSIAL untuk cluster Kafka Anda. IAM dan IAM dengan SASL/OAUTHBEARER otentikasi tidak menggunakan rahasia.

k. (Opsional) Untuk Enkripsi, pilih rahasia Manajer Rahasia yang berisi sertifikat CA root yang digunakan broker Kafka Anda untuk enkripsi TLS, jika broker Kafka Anda menggunakan sertifikat yang ditandatangani oleh CA pribadi.

Pengaturan ini berlaku untuk enkripsi TLS untuk SASL/SCRAM atau SASL/PLAIN, dan untuk otentikasi MTL.

l. Untuk membuat pemicu dalam status nonaktif untuk pengujian (disarankan), hapus Aktifkan pemicu. Atau, untuk segera mengaktifkan pemicu, pilih Aktifkan pemicu.

5. Untuk membuat pemicu, pilih Tambahkan.

Menambahkan cluster Kafka yang dikelola sendiri (AWS CLI)

Gunakan contoh berikut AWS CLI perintah untuk membuat dan melihat pemicu Apache Kafka yang dikelola sendiri untuk Lambda Event Function Anda.

Menggunakan SASL/SCRAM

Jika pengguna Kafka mengakses broker Kafka Anda melalui internet, tentukan rahasia Manajer Rahasia yang Anda buat SASL/SCRAM otentikasi. Contoh berikut menggunakan buat-pemetaan sumber-peristiwa AWS CLI perintah untuk memetakan Fungsi Peristiwa Lambda bernama `my-kafka-function` ke topik Kafka bernama `AWSKafkaTopic`.

```
aws lambda create-event-source-mapping \
  --topics AWSKafkaTopic \
  --source-access-configuration Type=SASL_SCRAM_512_AUTH,URI=arn:aws:secretsmanager:us-east-1:111122223333:secret:MyBrokerSecretName \
  --function-name arn:aws:lambda:us-east-1:111122223333:function:my-kafka-function \
  --self-managed-event-source '{"Endpoints":{"KAFKA_BOOTSTRAP_SERVERS":["abc3.xyz.com:9092", "abc2.xyz.com:9092"]}}'
```

Menggunakan VPC

Jika hanya Kafka pengguna dalam VPC Anda yang mengakses broker Kafka Anda, Anda harus menentukan VPC, subnet, dan grup keamanan VPC Anda. Contoh berikut menggunakan buat-pemetaan sumber-peristiwa AWS CLI perintah untuk memetakan Fungsi Peristiwa Lambda bernama `my-kafka-function` ke topik Kafka bernama `AWSKafkaTopic`.

```
aws lambda create-event-source-mapping \
  --topics AWSKafkaTopic \
  --source-access-configuration '[{"Type": "VPC_SUBNET", "URI":
 "subnet:subnet-0011001100"}, {"Type": "VPC_SUBNET", "URI":
 "subnet:subnet-0022002200"}, {"Type": "VPC_SECURITY_GROUP", "URI":
 "security_group:sg-0123456789"}]' \
```

```
  --function-name arn:aws:lambda:us-east-1:111122223333:function:my-kafka-function \
  --self-managed-event-source '{"Endpoints":{"KAFKA_BOOTSTRAP_SERVERS":
["abc3.xyz.com:9092", "abc2.xyz.com:9092"]}}'
```

Melihat status menggunakan AWS CLI

Contoh berikut menggunakan dapatkan-pemetaan sumber-acara AWS CLI perintah untuk menggambarkan status pemetaan sumber peristiwa yang Anda buat.

```
aws lambda get-event-source-mapping
              --uuid dh38738e-992b-343a-1077-3478934hjkfd7
```

Semua parameter konfigurasi sumber peristiwa Apache Kafka yang dikelola sendiri di Lambda

Semua jenis sumber peristiwa Lambda berbagi hal yang sama CreateEventSourceMapping dan UpdateEventSourceMapping Operasi API. Namun, hanya beberapa parameter yang berlaku untuk Apache Kafka yang dikelola sendiri, seperti yang ditunjukkan pada tabel berikut.

Parameter	Diperlukan	Default	Catatan
BatchSize	T	100	Maksimum: 10.000.
SelfManagedKafkaEventSourceConfig.ConsumptionMode	T	Streaming	`Stream` atau `Queue`. Menentukan bagaimana Lambda mengkonsumsi catatan dari topik. Dapat diatur hanya pada Create. Kapan `Queue`, `StartingPosition` , `StartingPositionTimestamp` , dan `MaximumBatchingWindowInSeconds` jangan berlaku.

Parameter	Diperlukan	Default	Catatan
			Lihat perinciannya di Polling topik Kafka dalam mode Antrian.
DestinationConfig	T	N/A	the section called “Pertahankan pemanggilan yang gagal”
Diaktifkan	T	True	
FilterCriteria	T	N/A	Kontrol peristiwa mana yang dikirim Lambda ke Fungsi Acara Anda
FunctionName	Y	N/A	
KMSKeyArn	T	N/A	the section called “Enkripsi kriteria filter”
MaximumBatchingWindowInSeconds	T	500 ms	Perilaku batching Tidak berlaku ketika `ConsumptionMode` adalah `Queue`.
ProvisionedPollersConfig	T	`MinimumPollers` : nilai default 1 jika tidak ditentukan `MaximumPollers` : nilai default 200 jika tidak ditentukan `PollerGroupName` : N/A	the section called “Mode yang disediakan”

Parameter	Diperlukan	Default	Catatan
SelfManagedEventSource	Y	N/A	Daftar Broker Kafka. Hanya dapat mengatur di Create
SelfManagedKafkaEventSourceConfig	T	Berisi ConsumerGroupId bidang yang default ke nilai unik.	Hanya dapat mengatur di Create
SourceAccessConfigurations	T	Tidak ada kredenensi	Informasi VPC atau kredensial autentikasi untuk klaster Untuk SASL_PLAIN, atur ke BASIC_AUT
StartingPosition	Y (Hanya mode Stream)	N/A	AT_TIMESTAMP, TRIM_HORIZON, atau TERBARU Hanya dapat mengatur di Create Tidak berlaku ketika `ConsumptionMode` adalah `Queue`.
StartingPositionTimestamp	T	N/A	Diperlukan jika StartingPosition diatur ke AT_TIMESTAMP Tidak berlaku ketika `ConsumptionMode` adalah `Queue`.
Tanda	T	N/A	the section called “Tag pemetaan sumber acara”

Parameter	Diperlukan	Default	Catatan
Topik	Y	N/A	Nama topik Hanya dapat mengatur di Create

> **Note**
>
> Ketika Anda menentukan a `PollerGroupName`, beberapa ESM dalam Amazon VPC yang sama dapat berbagi kapasitas Event Poller Unit (EPU). Anda dapat menggunakan opsi ini untuk mengoptimalkan biaya Mode yang disediakan untuk ESM Anda. Persyaratan untuk pengelompokan ESM:
>
> - ESM harus berada dalam Amazon VPC yang sama
> - Maksimal 100 ESM per grup poller
> - Agregat poller maksimum di semua ESM dalam grup tidak boleh melebihi 2000
>
> Anda dapat memperbarui `PollerGroupName` untuk memindahkan ESM ke grup yang berbeda, atau menghapus ESM dari grup dengan mengatur `PollerGroupName` ke string kosong (“”).

Mode penskalaan poller acara Apache Kafka di Lambda

Anda dapat memilih antara dua mode penskalaan poller acara untuk Amazon MSK dan pemetaan sumber acara Apache Kafka yang dikelola sendiri:

- On-demand modus (default)
- Mode yang disediakan

On-demand modus (default)

Saat Anda awalnya membuat sumber acara Kafka, Lambda mengalokasikan sejumlah poller acara default untuk memproses semua partisi dalam topik Kafka. Lambda secara otomatis menaikkan atau menurunkan jumlah poller acara berdasarkan pemuatan pesan.

Dalam interval satu menit, Lambda mengevaluasi lag offset dari semua partisi dalam topik. Jika offset lag terlalu tinggi, partisi menerima pesan lebih cepat daripada Lambda dapat memprosesnya. Jika perlu, Lambda menambahkan atau menghapus poller acara dari topik. Proses penskalaan otomatis untuk menambah atau menghapus poller peristiwa ini terjadi dalam waktu tiga menit setelah evaluasi.

Jika Fungsi Peristiwa Lambda target Anda dibatasi, Lambda mengurangi jumlah poller acara. Tindakan ini mengurangi beban kerja pada fungsi dengan mengurangi jumlah pesan yang dapat diambil dan dikirim oleh poller peristiwa ke fungsi.

Mode yang disediakan

Untuk beban kerja di mana Anda perlu menyempurnakan throughput pemetaan sumber peristiwa, Anda dapat menggunakan mode yang disediakan. Dalam mode yang disediakan, Anda menentukan batas minimum dan maksimum untuk jumlah poller acara yang disediakan. Poller acara yang disediakan ini didedikasikan untuk pemetaan sumber acara Anda, dan dapat menangani lonjakan pesan yang tidak terduga melalui penskalaan otomatis responsif. Kami menyarankan Anda menggunakan mode yang disediakan untuk beban kerja Kafka yang memiliki persyaratan kinerja yang ketat.

Di Lambda, poller peristiwa adalah unit komputasi dengan kemampuan throughput yang bervariasi menurut jenis sumber peristiwa. Untuk Amazon MSK dan Apache Kafka yang dikelola sendiri, setiap poller acara dapat menangani hingga 5 throughput atau hingga 5 MB/sec pemanggilan bersamaan. Misalnya, jika sumber acara Anda menghasilkan payload rata-rata 1 MB dan durasi rata-rata fungsi Anda adalah 1 detik, satu poller acara Kafka dapat mendukung 5 MB/sec throughput dan 5 pemanggilan Lambda bersamaan (dengan asumsi tidak ada transformasi payload). Untuk

Amazon SQS, setiap poller peristiwa dapat menangani hingga 1 throughput atau hingga MB/sec 10 pemanggilan bersamaan. Menggunakan mode yang disediakan menimbulkan biaya tambahan berdasarkan penggunaan poller acara Anda. Untuk detail harganya, lihat Harga AWS Lambda.

> **Note**
>
> Saat menggunakan mode yang disediakan, Anda tidak perlu membuat titik akhir AWS PrivateLink VPC atau memberikan izin terkait sebagai bagian dari konfigurasi jaringan Anda.

Dalam mode yang disediakan, kisaran nilai yang diterima untuk jumlah minimum poller acara (`MinimumPollers`) adalah antara 1 dan 200, inklusif. Kisaran nilai yang diterima untuk jumlah maksimum poller acara (`MaximumPollers`) adalah antara 1 dan 2.000, inklusif. `MaximumPollers`harus lebih besar dari atau sama dengan`MinimumPollers`. Selain itu, untuk mempertahankan pemrosesan yang teratur dalam partisi, Lambda membatasi `MaximumPollers` jumlah partisi dalam topik.

Untuk detail selengkapnya tentang memilih nilai yang sesuai untuk poller acara minimum dan maksimum, lihatPraktik terbaik.

Anda dapat mengonfigurasi mode yang disediakan untuk pemetaan sumber acara Kafka menggunakan konsol atau API Lambda.

Untuk mengonfigurasi mode yang disediakan untuk pemetaan sumber peristiwa yang ada (konsol)

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih fungsi dengan pemetaan sumber peristiwa yang ingin Anda konfigurasikan untuk mode yang disediakan.
3. Pilih Konfigurasi, lalu pilih Pemicu.
4. Pilih pemetaan sumber peristiwa yang ingin Anda konfigurasikan untuk mode yang disediakan, lalu pilih Edit.
5. Di bawah Mode yang disediakan, pilih Konfigurasi.

    - Untuk poller acara Minimum, masukkan nilai antara 1 dan 200. Jika Anda tidak menentukan nilai, Lambda memilih nilai default 1.
    - Untuk poller acara Maksimum, masukkan nilai antara 1 dan 2.000. Nilai ini harus lebih besar dari atau sama dengan nilai Anda untuk poller acara Minimum. Jika Anda tidak menentukan nilai, Lambda memilih nilai default 200.

6. Pilih Simpan.

Anda dapat mengonfigurasi mode yang disediakan secara terprogram menggunakan objek di. ProvisionedPollerConfig EventSourceMappingConfiguration Misalnya, perintah UpdateEventSourceMappingCLI berikut mengkonfigurasi `MinimumPollers` nilai 5, dan `MaximumPollers` nilai 100.

```
aws lambda update-event-source-mapping \
    --uuid a1b2c3d4-5678-90ab-cdef-EXAMPLE11111 \
    --provisioned-poller-config '{"MinimumPollers": 5, "MaximumPollers": 100}'
```

Setelah mengonfigurasi mode yang disediakan, Anda dapat mengamati penggunaan poller peristiwa untuk beban kerja Anda dengan memantau metrik. `ProvisionedPollers` Untuk informasi selengkapnya, lihat the section called “Metrik pemetaan sumber acara”.

Untuk menonaktifkan mode yang disediakan dan kembali ke mode default (sesuai permintaan), Anda dapat menggunakan perintah CLI UpdateEventSourceMappingberikut:

```
aws lambda update-event-source-mapping \
    --uuid a1b2c3d4-5678-90ab-cdef-EXAMPLE11111 \
    --provisioned-poller-config '{}'
```

Fitur penanganan kesalahan dan kinerja tingkat lanjut

Untuk pemetaan sumber peristiwa Kafka dengan mode yang disediakan diaktifkan, Anda dapat mengonfigurasi fitur tambahan untuk meningkatkan penanganan dan kinerja kesalahan:

- Konfigurasi coba lagi — Kontrol cara Lambda menangani catatan yang gagal dengan upaya coba ulang maksimum, batas usia rekaman, pemisahan batch, dan respons batch sebagian.
- Tujuan kegagalan Kafka — Kirim catatan yang gagal ke topik Kafka untuk diproses atau dianalisis nanti.

Praktik dan pertimbangan terbaik saat menggunakan mode yang disediakan

Konfigurasi optimal poller peristiwa minimum dan maksimum untuk pemetaan sumber acara Anda bergantung pada persyaratan kinerja aplikasi Anda. Kami menyarankan Anda memulai dengan poller acara minimum default untuk mendasarkan profil kinerja. Sesuaikan konfigurasi Anda berdasarkan pola pemrosesan pesan yang diamati dan profil kinerja yang Anda inginkan.

Untuk beban kerja dengan lalu lintas runcing dan kebutuhan kinerja yang ketat, tingkatkan poller acara minimum untuk menangani lonjakan pesan yang tiba-tiba. Untuk menentukan poller kejadian minimum yang diperlukan, pertimbangkan pesan beban kerja Anda per detik dan ukuran muatan rata-rata, dan gunakan kapasitas throughput dari poller peristiwa tunggal (hingga 5 MBps) sebagai referensi.

Untuk mempertahankan pemrosesan yang teratur dalam partisi, Lambda membatasi poller peristiwa maksimum ke jumlah partisi dalam topik. Selain itu, poller peristiwa maksimum yang dapat diskalakan oleh pemetaan sumber acara Anda bergantung pada pengaturan konkurensi fungsi.

Saat mengaktifkan mode yang disediakan, perbarui pengaturan jaringan Anda untuk menghapus titik akhir AWS PrivateLink VPC dan izin terkait.

Optimalisasi Biaya untuk mode Provisioned

Harga mode yang disediakan

Mode yang disediakan dibebankan berdasarkan poller peristiwa minimum yang disediakan, dan poller acara dikonsumsi selama penskalaan otomatis. Biaya dihitung menggunakan unit penagihan yang disebut Event Poller Unit (EPU). Anda membayar untuk jumlah dan durasi EPU yang digunakan, diukur dalam Event-Poller-Unit-hours.

Anda dapat menggunakan mode Provisioned dengan satu ESM untuk aplikasi yang peka terhadap kinerja atau Anda dapat mengelompokkan beberapa ESM dalam VPC yang sama untuk berbagi kapasitas dan biaya EPU. Bagian berikut menjelaskan dua kemampuan yang membantu Anda mengoptimalkan biaya mode Provisioned Anda. Untuk detail harga, lihat harga AWS Lambda.

Peningkatan Pemanfaatan EPU

Setiap EPU mendukung kapasitas MB/s throughput hingga 20 untuk polling acara dan mendukung default 10 poller acara. Saat Anda membuat mode Provisioned untuk Kafka ESM dengan menyetel poller minimum dan maksimum, mode ini menggunakan nomor poller minimum untuk menyediakan EPU berdasarkan default 10 poller acara per EPU. Namun, setiap poller acara dapat menskalakan secara independen untuk mendukung hingga 5 MB/s kapasitas throughput, yang mungkin memerlukan kepadatan poller peristiwa yang lebih rendah pada EPU tertentu dan dapat memicu penskalaan EPU. Jumlah poller acara yang dialokasikan pada EPU tergantung pada kapasitas komputasi yang dikonsumsi oleh masing-masing poller acara. Pendekatan pemanfaatan EPU yang ditingkatkan ini memungkinkan poller acara dengan berbagai persyaratan throughput untuk menggunakan kapasitas EPU secara efektif, mengurangi biaya untuk semua ESM.

Pengelompokan ESM

Untuk mengoptimalkan biaya mode Provisioned Anda lebih lanjut, Anda dapat mengelompokkan beberapa ESM Kafka untuk berbagi kapasitas EPU. Dengan pengelompokan ESM dan Pemanfaatan EPU yang ditingkatkan, Anda dapat mengurangi biaya mode Provisioned hingga 90% untuk beban kerja throughput rendah dibandingkan dengan berjalan dalam mode ESM tunggal. Semua ESM yang membutuhkan kurang dari 1 kapasitas EPU mendapat manfaat dari pengelompokan ESM. ESM semacam itu biasanya membutuhkan beberapa poller acara minimum untuk mendukung kebutuhan throughputnya. Dengan kemampuan ini, Anda dapat mengadopsi mode Provisioned untuk semua beban kerja Kafka Anda, dan mendapatkan manfaat dari fitur seperti validasi skema, pemfilteran Avro/Protobuf peristiwa, pemanggilan latensi rendah, dan penanganan kesalahan yang disempurnakan yang hanya tersedia dalam mode Provisioned.

Saat Anda mengonfigurasi `PollerGroupName` parameter dengan nilai yang sama untuk beberapa ESM dalam VPC Amazon yang sama, ESM tersebut berbagi sumber daya EPU alih-alih masing-masing memerlukan kapasitas EPU khusus. Anda dapat mengelompokkan hingga 100 ESM per grup poller, dan agregat poller maksimum di semua ESM di satu grup tidak boleh lebih dari 2000.

Untuk mengkonfigurasi pengelompokan ESM (konsol)

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih fungsi Anda.
3. Pilih Konfigurasi, lalu pilih Pemicu.
4. Saat membuat pemetaan sumber peristiwa Kafka baru, atau mengedit yang sudah ada, pilih Konfigurasi di bawah Mode yang disediakan.
5. Untuk poller acara Minimum, masukkan nilai antara 1 dan 200.
6. Untuk poller acara Maksimum, masukkan nilai antara 1 dan 2.000.
7. Untuk nama grup Poller, masukkan pengenal untuk grup. Gunakan nama yang sama untuk ESM lain yang ingin Anda kelompokkan bersama.
8. Pilih Simpan.

Untuk mengkonfigurasi pengelompokan ESM (AWS CLI)

Contoh berikut membuat ESM dengan grup poller bernama: `production-app-group`

```
aws lambda create-event-source-mapping \
  --function-name myFunction1 \
```

```
  --event-source-arn arn:aws:kafka:us-east-1:123456789012:cluster/MyCluster/abcd1234 \
  --topics topic1 \
  --starting-position LATEST \
  --provisioned-poller-config '{
    "MinimumPollers": 1,
    "MaximumPollers": 10,
    "PollerGroupName": "production-app-group"
  }'
```

Untuk menambahkan ESM lain ke grup yang sama (berbagi kapasitas EPU), gunakan yang sama: PollerGroupName

```
aws lambda create-event-source-mapping \
  --function-name myFunction2 \
  --event-source-arn arn:aws:kafka:us-east-1:123456789012:cluster/MyCluster/abcd1234 \
  --topics topic2 \
  --starting-position LATEST \
  --provisioned-poller-config '{
    "MinimumPollers": 1,
    "MaximumPollers": 10,
    "PollerGroupName": "production-app-group"
  }'
```

> **Note**
>
> Anda dapat memperbarui `PollerGroupName` untuk memindahkan ESM ke grup lain, atau menghapus ESM dari grup dengan meneruskan string kosong (“”) untuk: `PollerGroupName`

```
# Move ESM to a different group
aws lambda update-event-source-mapping \
  --uuid a1b2c3d4-5678-90ab-cdef-EXAMPLE11111 \
  --provisioned-poller-config '{
    "MinimumPollers": 1,
    "MaximumPollers": 10,
    "PollerGroupName": "new-group-name"
  }'

# Remove ESM from group (use dedicated resources)
aws lambda update-event-source-mapping \
  --uuid a1b2c3d4-5678-90ab-cdef-EXAMPLE11111 \
```

```
  --provisioned-poller-config '{
    "MinimumPollers": 1,
    "MaximumPollers": 10,
    "PollerGroupName": ""
  }'
```

Pertimbangan strategi pengelompokan

- Batas aplikasi — Grup ESM yang termasuk dalam aplikasi atau layanan yang sama untuk alokasi dan manajemen biaya yang lebih baik. Pertimbangkan untuk menggunakan konvensi penamaan seperti `app-name-environment` (misalnya,`order-processor-prod`).
- Pola lalu lintas — Hindari pengelompokan ESM dengan throughput tinggi dan pola lalu lintas runcing, karena ini dapat menyebabkan pertengkaran sumber daya.
- Radius ledakan — Pertimbangkan dampaknya jika infrastruktur bersama mengalami masalah. Semua ESM dalam grup yang sama dipengaruhi oleh keterbatasan sumber daya bersama. Untuk beban kerja penting misi, Anda mungkin ingin menggunakan grup terpisah atau ESM khusus.

Contoh optimasi biaya

Pertimbangkan skenario di mana Anda memiliki 10 ESM, masing-masing dikonfigurasi dengan 1 poller peristiwa dan throughput di bawah 2: MB/s

Tanpa pengelompokan:

- Setiap ESM membutuhkan EPU-nya sendiri
- Total EPU yang dibutuhkan: 10
- Biaya per EPU: $0. 185/hour di AS Timur (Virginia N.)
- Biaya EPU bulanan (720 jam): 10 × 720 × $0,185 = $1,332

Dengan pengelompokan:

- Semua 10 ESM berbagi kapasitas EPU
- 10 poller acara cocok dalam 1 EPU (dengan 10 poller baru per dukungan EPU)
- Total EPU yang dibutuhkan: 1
- Biaya EPU bulanan (720 jam): 1 × 720 × $0.185 = $133.20
- Penghematan biaya: 90% (penghematan $1,198.80 per bulan)

Polling Apache Kafka dan posisi awal streaming di Lambda

StartingPosition Parameter memberi tahu Lambda kapan harus mulai membaca pesan dari Amazon MSK atau aliran Apache Kafka yang dikelola sendiri. Ada tiga opsi untuk dipilih:

- Ter baru — Lambda mulai membaca tepat setelah catatan terbaru dalam topik Kafka.
- Trim horizon — Lambda mulai membaca dari catatan terakhir yang belum dipangkas dalam topik Kafka. Ini juga catatan tertua dalam topik.
- Pada stempel waktu — Lambda mulai membaca dari posisi yang ditentukan oleh stempel waktu, dalam detik waktu Unix. Gunakan StartingPositionTimestamp parameter untuk menentukan stempel waktu.

Polling aliran selama pembuatan atau pembaruan pemetaan sumber peristiwa pada akhirnya konsisten:

- Selama pembuatan pemetaan sumber peristiwa, mungkin diperlukan beberapa menit untuk memulai polling peristiwa dari aliran.
- Selama pembaruan pemetaan sumber peristiwa, mungkin diperlukan waktu hingga 90 detik untuk menghentikan dan memulai ulang peristiwa polling dari aliran.

Perilaku ini berarti bahwa jika Anda menentukan `LATEST` sebagai posisi awal untuk aliran, pemetaan sumber peristiwa dapat melewatkan peristiwa selama pembuatan atau pembaruan. Untuk memastikan bahwa tidak ada acara yang terlewatkan, tentukan salah `TRIM_HORIZON` satu atau`AT_TIMESTAMP`.

Polling topik Kafka dalam mode Antrian

Mode antrian adalah mode konsumsi yang menggunakan grup berbagi, yang tersedia di Apache Kafka 4.0 (Kafka Queue Poller memerlukan versi 4.2 atau yang lebih baru). Dengan mode Antrian, Anda dapat memproses pesan topik Kafka dengan cara throughput tinggi yang tidak teratur menggunakan grup berbagi. Tidak seperti kelompok konsumen Kafka tradisional yang menetapkan partisi secara eksklusif untuk konsumen, grup berbagi memungkinkan beberapa konsumen membaca dari partisi yang sama secara bersamaan. Ini mirip dengan cara Amazon SQS memproses pesan dari antrian.

Saat Anda membuat pemetaan sumber peristiwa untuk pengelolaan sendiri Apache Kafka cluster, Anda dapat memilih antara dua mode konsumsi:

- Streaming (default). Menggunakan kelompok konsumen Kafka dengan pemesanan per partisi. Lambda menetapkan setiap partisi ke satu konsumen.
- Antrian. Menggunakan grup berbagi, yang tersedia di Apache Kafka 4.0, untuk pemrosesan throughput tinggi yang tidak teratur (Kafka Queue Poller memerlukan versi 4.2 atau lebih baru). Dengan grup berbagi, beberapa konsumen dapat membaca dari partisi yang sama secara bersamaan, mirip dengan cara Amazon SQS memproses pesan dari antrian.

Jika Anda tidak menentukan mode konsumsi, pemetaan sumber peristiwa menggunakan Stream. Halaman ini mencakup mode konsumsi antrian. Untuk informasi umum tentang menggunakan Lambda dengan self-managed Apache Kafka, lihat Menggunakan Lambda dengan Apache Kafka yang dikelola sendiri.

Dalam mode Antrian, Lambda menggunakan Kafka Share Consumer untuk polling pesan dari topik. Setiap poller peristiwa memproses pesan di semua partisi tanpa pemesanan tingkat partisi. Ini meningkatkan konkurensi untuk setiap poller.

Persyaratan minimum

Untuk menggunakan mode konsumsi antrian, cluster Kafka Anda harus memenuhi persyaratan berikut:

- Apache Kafka versi 4.2 atau lebih baru dengan grup berbagi diaktifkan. Kafka memungkinkan `share.version` secara default, dan Anda dapat mengelolanya melalui `kafka-features.sh`.
- Self-managed Apache Kafka (SMK) cluster saja. Streaming Terkelola Amazon untuk Apache Kafka (Amazon MSK) saat ini tidak mendukung mode Antrian. AWS menambahkan dukungan MSK setelah MSK menawarkan Kafka 4.2 atau lebih baru dengan grup berbagi diaktifkan.

Perbandingan Streaming dan Antrian

Fitur	Streaming (default)	Antrean
Pemesanan	Per-partition pemesanan dijamin	Tidak ada jaminan pemesanan
Model konsumen	Kelompok konsumen (penugasan partisi eksklusif)	Bagikan grup (akses partisi bersamaan)
Throughput per event poller	Hingga 5 MB/s	Hingga 5 MB/s
Pemanggilan bersamaan per poller peristiwa	Hingga 5	Hingga 10
Mode penskalaan	On-demand dan disediakan	On-demand dan disediakan
Berbagi EPU	Dapat berbagi EPU dengan pemetaan sumber acara Kafka Streams lainnya	Terisolasi. Pemetaan sumber peristiwa antrian hanya dibagikan dengan pemetaan sumber peristiwa Antrian lainnya
Kasus penggunaan	Pemrosesan acara yang dipesan	High-throughput beban kerja yang tidak teratur

Buat pemetaan antrian

Untuk membuat pemetaan sumber peristiwa mode Antrian, atur `ConsumptionMode` bidang `SelfManagedKafkaEventSourceConfig` kepada `Queue`. Dalam contoh berikut, ganti `my-function`, titik akhir broker, dan `my-queue-topic` dengan nilai-nilai Anda sendiri.

AWS CLI

Berikut ini AWS Perintah CLI membuat pemetaan sumber peristiwa mode antrian. Ganti *`my-function`* dengan nama fungsi Anda, titik akhir broker dengan broker Anda, dan *`my-queue-topic`* dengan topik Anda.

```
PROMPT> aws lambda create-event-source-mapping \
```

```
  --function-name my-function \
  --self-managed-event-source '{"Endpoints":{"KAFKA_BOOTSTRAP_SERVERS":["broker1.us-
west-2.example.com:9092","broker2.us-west-2.example.com:9092"]}}' \
  --topics my-queue-topic \
  --batch-size 100 \
  --self-managed-kafka-event-source-config '{"ConsumptionMode": "Queue"}' \
  --provisioned-poller-config '{"MinimumPollers": 1, "MaximumPollers": 10}'
```

AWS CloudFormation

Berikut ini AWS CloudFormation template membuat pemetaan sumber peristiwa mode antrian. Ganti titik akhir broker dan *my-queue-topic* dengan nilai-nilai Anda sendiri.

```
MyKafkaQueueESM:
  Type: AWS::Lambda::EventSourceMapping
  Properties:
    FunctionName: !Ref MyFunction
    SelfManagedEventSource:
      Endpoints:
        KafkaBootstrapServers:
          - broker1.us-west-2.example.com:9092
          - broker2.us-west-2.example.com:9092
    Topics:
      - my-queue-topic
    BatchSize: 100
    SelfManagedKafkaEventSourceConfig:
      ConsumptionMode: Queue
    ProvisionedPollerConfig:
      MinimumPollers: 1
      MaximumPollers: 10
```

Parameter konfigurasi

Tabel berikut mencantumkan parameter yang berlaku untuk pemetaan sumber peristiwa mode antrian.

Parameter	Deskripsi	Rentang/Default
`BatchSize`	Jumlah maksimum catatan per batch yang dikirim ke fungsi	1—10.000. Default: 100

Parameter	Deskripsi	Rentang/Default
`Enabled`	Apakah pemetaan sumber peristiwa aktif	`true` atau `false`. Default: `true`
`FilterCriteria`	Pola pemfilteran peristiwa untuk memproses hanya catatan yang cocok	Hingga 5 pola filter
`MaximumRetryAttempts`	Percobaan ulang maksimum untuk catatan yang gagal	-1-10.000. Default: 0 (dalam mode Antrian, broker Kafka melakukan percobaan ulang ketika tidak ada tujuan pada kegagalan yang dikonfigurasi)
`BisectBatchOnFunctionError`	Pisahkan batch menjadi dua pada kesalahan fungsi	`true` atau `false`. Default: `false`
`ProvisionedPollerConfig.MinimumPollers`	Poller acara minimum (mode disediakan)	1—200. Default: 1
`ProvisionedPollerConfig.MaximumPollers`	Poller peristiwa maksimum (mode disediakan)	1—2.000. Standar: 200
`ProvisionedPollerConfig.PollerGroupName`	Nama grup untuk berbagi EPU (mode disediakan, hanya pemetaan sumber acara antrian)	String. Hingga 100 pemetaan sumber peristiwa per grup
`SelfManagedKafkaEventSourceConfig.ConsumptionMode`	Mode konsumsi untuk sumber acara	Harus `Queue` untuk mode Antrian
`SchemaRegistryConfig`	Registri skema untuk validasi acara	Lihat Menggunakan pendaftar skema dengan sumber acara Kafka di Lambda

> **Important**
>
> `MaximumBatchingWindowInSeconds` Parameter adalah tidak didukung untuk pemetaan sumber peristiwa mode antrian. Konsumen Saham Kafka menggunakan pengakuan eksplisit. Semua catatan dalam batch harus diakui secara eksplisit sebelum konsumen dapat mengambil yang baru. Ini tidak kompatibel dengan jendela batching berbasis waktu. Jika Anda menentukan `MaximumBatchingWindowInSeconds` saat membuat pemetaan sumber peristiwa mode Antrian, API mengembalikan kesalahan validasi.

Fitur pemetaan sumber acara yang didukung

Tabel berikut mencantumkan fitur pemetaan sumber peristiwa Lambda yang tersedia dengan mode Antrian.

Fitur	Didukung	Catatan
Mode yang disediakan	Ya	Konfigurasikan `MinimumPollers` dan `MaximumPollers`
On-demand mode	Ya	Mode penskalaan default. Lambda menskalakan poller secara otomatis hingga 5 MB/s atau 50 pemanggilan bersamaan, mana yang dicapai lebih dulu. Tidak diperlukan konfigurasi tambahan.
Pemfilteran peristiwa	Ya	Hingga 5 pola filter per pemetaan sumber peristiwa
Skema registri	Ya	Avro dan Protobuf dengan Glue Schema Registry atau Confluent Schema Registry. Pemfilteran registri skema

Fitur	Didukung	Catatan
		hanya tersedia dalam mode yang disediakan.
Low-latency pengolahan	Ya	Tersedia dalam mode disediakan
Coba lagi konfigurasi	Ya	`MaximumRetryAttempts` , `BisectBatchOnFunctionError`
On-failure tujuan (topik Kafka)	Ya	Mengirim catatan yang gagal ke topik Kafka
On-failure tujuan (SNS/SQS)	Ya	Kirim catatan yang gagal ke SNS atau SQS
Pencatatan pemetaan sumber peristiwa	Ya	CloudWatch Log untuk diagnostik pemetaan sumber peristiwa
Pengelompokan pemetaan sumber acara (`PollerGroupName`)	Ya	Pemetaan sumber peristiwa mode antrian dapat berbagi EPU dengan pemetaan sumber peristiwa mode antrian lainnya saja (mode disediakan)
Jendela batch	Tidak	Tidak didukung karena perilaku mode pemblokiran Share Consumer
Posisi awal	Tidak	`StartingPosition` dan `StartingPositionTimestamp` tidak didukung. Mode antrian selalu dibaca dari posisi berkomitmen grup berbagi.

Mode penskalaan

Pemetaan sumber peristiwa mode antrian mendukung mode penskalaan sesuai permintaan (default) dan yang disediakan.

On-demand mode (default)

Dalam mode on-demand, Lambda secara otomatis mengelola poller peristiwa untuk pemetaan sumber peristiwa mode Antrian Anda.

On-demand mode menskalakan hingga 5 poller peristiwa per pemetaan sumber peristiwa. Mencapai plafonnya di 5 MB/s throughput agregat atau 50 pemanggilan bersamaan (5 poller dikalikan 10). Ini terjadi batas mana pun yang tercapai lebih dulu. Untuk menskalakan melampaui batas ini, gunakan mode yang disediakan. On-demand mode tidak memerlukan konfigurasi tambahan di luar parameter pemetaan sumber peristiwa standar.

Tabel berikut mencantumkan batas mode sesuai permintaan.

Kuota	Nilai
Throughput per event poller	Hingga 1 MB/s
Pemanggilan bersamaan per poller peristiwa	Hingga 10
Poller acara maksimum	5
Throughput agregat maksimum	Hingga 5 MB/s (5 poller kali 1 MB/s)
Konkurensi agregat maksimum	Hingga 50 pemanggilan bersamaan (5 poller kali 10)

Contoh - Pemetaan sumber peristiwa mode antrian dalam mode on-demand (AWS CLI)

Berikut ini AWS Perintah CLI membuat pemetaan sumber peristiwa mode Antrian dalam mode on-demand. Ganti *my-function*, titik akhir broker, *my-queue-topic*, ID subnet, dan ID grup keamanan dengan nilai Anda sendiri.

```
PROMPT> aws lambda create-event-source-mapping \
  --function-name my-function \
```

```
  --self-managed-event-source '{"Endpoints":{"KAFKA_BOOTSTRAP_SERVERS":["broker1.us-
west-2.example.com:9092","broker2.us-west-2.example.com:9092"]}}' \
  --topics my-queue-topic \
  --batch-size 100 \
  --self-managed-kafka-event-source-config '{"ConsumptionMode": "Queue"}' \
  --source-access-configurations
 '[{"Type":"VPC_SUBNET","URI":"subnet-0123456789abcdef0"},
{"Type":"VPC_SUBNET","URI":"subnet-0123456789abcdef1"},
{"Type":"VPC_SECURITY_GROUP","URI":"sg-0123456789abcdef0"}]'
```

Mode yang disediakan

Untuk beban kerja dengan persyaratan kinerja yang ketat atau pola lalu lintas yang runcing, gunakan mode yang disediakan. Mode yang disediakan memberi Anda kontrol langsung atas jumlah poller peristiwa minimum dan maksimum yang didedikasikan untuk pemetaan sumber peristiwa mode Antrian Anda.

Setiap poller peristiwa mode Antrian dalam mode disediakan mendukung hingga 5 MB/s throughput dan hingga 10 pemanggilan Lambda bersamaan.

Tabel berikut mencantumkan batas mode yang disediakan.

Kuota	Nilai
Throughput per event poller	Hingga 5 MB/s
Pemanggilan bersamaan per poller peristiwa	Hingga 10
Poller acara minimum (`MinimumPollers`)	1—200. Default: 1
Poller acara maksimum (`MaximumPollers`)	1—2.000. Standar: 200
Pemetaan sumber peristiwa per grup poller	Hingga 100
Agregat poller maksimum per kelompok poller	2.000
Throughput per EPU (agregat)	Hingga 5 MB/s per poller; agregat tergantung pada kepadatan kemasan poller

Mode yang disediakan dikenakan biaya berdasarkan penggunaan Event Poller Unit (EPU). Untuk detail harga, lihat AWS Lambda Harga.

Contoh - Pemetaan sumber peristiwa mode antrian dengan mode disediakan (AWS CLI)

Berikut ini AWS Perintah CLI membuat pemetaan sumber peristiwa mode antrian dalam mode disediakan.

```
PROMPT> aws lambda create-event-source-mapping \
  --function-name my-function \
  --self-managed-event-source '{"Endpoints":{"KAFKA_BOOTSTRAP_SERVERS":["broker1.us-
west-2.example.com:9092","broker2.us-west-2.example.com:9092"]}}' \
  --topics my-queue-topic \
  --batch-size 100 \
  --self-managed-kafka-event-source-config '{"ConsumptionMode": "Queue"}' \
  --provisioned-poller-config '{"MinimumPollers": 5, "MaximumPollers": 100}' \
  --source-access-configurations
 '[{"Type":"VPC_SUBNET","URI":"subnet-0123456789abcdef0"},
{"Type":"VPC_SUBNET","URI":"subnet-0123456789abcdef1"},
{"Type":"VPC_SECURITY_GROUP","URI":"sg-0123456789abcdef0"}]'
```

Ganti placeholder berikut dengan nilai Anda sendiri:

- *subnet-0123456789abcdef0* dan *subnet-0123456789abcdef1*: ID subnet Anda.
- *sg-0123456789abcdef0*: ID grup keamanan Anda.
- `MinimumPollers` dan `MaximumPollers`: diatur sesuai dengan kebutuhan throughput Anda.

> **Note**
>
> Untuk beralih dari mode yang disediakan kembali ke mode sesuai permintaan, berikan tanda kosong `ProvisionedPollerConfig`:
>
> ```
> PROMPT> aws lambda update-event-source-mapping \
>   --uuid a1b2c3d4-5678-90ab-cdef-EXAMPLE11111 \
>   --provisioned-poller-config '{}'
> ```

Pemilihan mode penskalaan

Pertimbangan	On-demand	Disediakan
Pengaturan	Tidak ada konfigurasi tambahan	Konfigurasikan `MinimumPollers` dan `MaximumPollers`
Biaya	Tidak ada biaya EPU tambahan	Biaya EPU berdasarkan penggunaan per jam
Terbaik untuk	Low-concurrency beban kerja	Persyaratan latensi yang ketat, lalu lintas runcing
Pengelompokan pemetaan sumber acara (`PollerGroupName`)	Tidak didukung	Didukung. Bagikan EPU di seluruh pemetaan sumber peristiwa Antrian
Low-latency pengolahan	Tidak tersedia	Available
Penyaringan registri skema (Avro/Protobuf)	Tidak tersedia	Available

Isolasi berbagi EPU

Dalam mode yang disediakan, Anda dapat mengelompokkan beberapa pemetaan sumber peristiwa dalam VPC yang sama untuk berbagi kapasitas Event Poller Unit (EPU) dengan menetapkan kapasitas yang sama `PollerGroupName`. Pengelompokan pemetaan sumber peristiwa memungkinkan pemetaan sumber peristiwa throughput rendah dikemas ke EPU bersama alih-alih masing-masing memerlukan kapasitas khusus, mengurangi biaya mode disediakan. Lihat perinciannya di Optimalisasi Biaya untuk mode Provisioned.

Pemetaan sumber peristiwa mode antrian hanya berbagi EPU dengan pemetaan sumber peristiwa mode Antrian lainnya. SEBUAH `PollerGroupName` digunakan oleh pemetaan sumber peristiwa mode Antrian tidak dapat dibagikan dengan pemetaan sumber peristiwa Kafka Streams, dan sebaliknya. Jika Anda mengkonfigurasi a `PollerGroupName` sudah digunakan oleh mode konsumsi lainnya, `CreateEventSourceMapping` mengembalikan kesalahan validasi. Untuk menggunakan

pengelompokan pemetaan sumber peristiwa dengan mode Antrian, buat `PollerGroupName` didedikasikan untuk pemetaan sumber acara Antrian Anda.

Contoh peristiwa

Ketika Lambda memanggil fungsi Anda dengan sumber peristiwa mode Antrian, payload peristiwa identik dengan peristiwa Kafka standar. Setiap catatan mencakup topik, partisi, offset, stempel waktu, dan kunci dan nilai yang dikodekan base64:

```
{
  "eventSource": "SelfManagedKafka",
  "bootstrapServers": "broker1.example.com:9092,broker2.example.com:9092",
  "records": {
    "my-queue-topic-0": [
      {
        "topic": "my-queue-topic",
        "partition": 0,
        "offset": 15,
        "timestamp": 1545084650987,
        "timestampType": "CREATE_TIME",
        "key": "abcDEFghiJKLmnoPQRstuVWXyz1234==",
        "value": "SGVsbG8sIHRoaXMgaXMgYSB0ZXN0Lg==",
        "headers": [
          {
            "headerKey": [104, 101, 97, 100, 101, 114, 86, 97, 108, 117, 101]
          }
        ]
      }
    ],
    "my-queue-topic-3": [
      {
        "topic": "my-queue-topic",
        "partition": 3,
        "offset": 42,
        "timestamp": 1545084651000,
        "timestampType": "CREATE_TIME",
        "key": "YW5vdGhlcktleQ==",
        "value": "VGhpcyBpcyBhbm90aGVyIG1lc3NhZ2U=",
        "headers": []
      }
    ]
  }
}
```

Idempotensi diperlukan

Pemetaan sumber peristiwa lambda memproses setiap peristiwa setidaknya sekali, sehingga pemrosesan rekaman duplikat dapat terjadi. Karena mode Antrian tidak menjamin pemesanan, pemrosesan duplikat atau di luar urutan dapat menghasilkan hasil yang salah, seperti penghitungan ganda atau status tidak konsisten, jika kode fungsi Anda tidak berpotensi. Jadikan kode fungsi Anda idempoten.

Batasan

- Self-managed Kelompok-kelompok Kafka saja. Amazon MSK tidak didukung saat peluncuran.
- Apache Kafka 4.2 atau yang lebih baru diperlukan. Cluster harus mendukung protokol Share Consumer dengan grup berbagi diaktifkan.
- Jendela batching tidak didukung. `MaximumBatchingWindowInSeconds` parameter tidak dapat digunakan dengan pemetaan sumber peristiwa Kafka Queue.
- Posisi awal tidak didukung. `StartingPosition` dan `StartingPositionTimestamp` tidak dapat digunakan dengan pemetaan sumber peristiwa mode Antrian. Grup saham mengkonsumsi dari posisi offset yang dikomitmennya.
- Isolasi EPU. Pemetaan sumber peristiwa mode antrian tidak dapat berbagi EPU dengan pemetaan sumber peristiwa Kafka Streams. Gunakan terpisah `PollerGroupName` nilai-nilai.

ID grup konsumen yang dapat disesuaikan di Lambda

Saat menyiapkan Amazon MSK atau Apache Kafka yang dikelola sendiri sebagai sumber acara, Anda dapat menentukan ID grup konsumen. ID grup konsumen ini adalah pengenal yang ada untuk grup konsumen Kafka yang Anda inginkan untuk bergabung dengan Fungsi Acara Lambda Anda. Anda dapat menggunakan fitur ini untuk memigrasikan pengaturan pemrosesan catatan Kafka yang sedang berlangsung dengan mulus dari konsumen lain ke Lambda.

Kafka mendistribusikan pesan ke semua konsumen dalam kelompok konsumen. Jika Anda menentukan ID grup konsumen yang memiliki konsumen aktif lainnya, Lambda hanya menerima sebagian pesan dari topik Kafka. Jika Anda ingin Lambda menangani semua pesan dalam topik, matikan konsumen lain di grup konsumen tersebut.

Selain itu, jika Anda menentukan ID grup konsumen, dan Kafka menemukan grup konsumen yang sudah ada yang valid dengan ID yang sama, Lambda mengabaikan StartingPositionpemetaan sumber acara Anda. Sebaliknya, Lambda mulai memproses catatan sesuai dengan offset yang dilakukan dari kelompok konsumen. Jika Anda menentukan ID grup konsumen, dan Kafka tidak dapat menemukan grup konsumen yang ada, maka Lambda mengonfigurasi sumber acara Anda dengan yang ditentukan. `StartingPosition`

ID grup konsumen yang Anda tentukan harus unik di antara semua sumber acara Kafka Anda. Setelah membuat pemetaan sumber acara Kafka dengan ID grup konsumen yang ditentukan, Anda tidak dapat memperbarui nilai ini.

Memfilter acara dari Amazon MSK dan sumber acara Apache Kafka yang dikelola sendiri

Anda dapat menggunakan pemfilteran peristiwa untuk mengontrol rekaman mana dari aliran atau antrian yang dikirim Lambda ke Fungsi Acara Anda. Untuk informasi umum tentang cara kerja penyaringan acara, lihatthe section called “Pemfilteran peristiwa”.

> **Note**
>
> Amazon MSK dan pemetaan sumber acara Apache Kafka yang dikelola sendiri hanya mendukung pemfilteran pada kunci. `value`

Topik

- Dasar-dasar penyaringan acara Kafka

Dasar-dasar penyaringan acara Kafka

Misalkan produser menulis pesan ke topik di cluster Kafka Anda, baik dalam format JSON yang valid atau sebagai string biasa. Contoh catatan akan terlihat seperti berikut, dengan pesan dikonversi ke string yang dikodekan Base64 di bidang. `value`

```
{
    "mytopic-0":[
        {
            "topic":"mytopic",
            "partition":0,
            "offset":15,
            "timestamp":1545084650987,
            "timestampType":"CREATE_TIME",
            "value":"SGVsbG8sIHRoaXMgaXMgYSB0ZXN0Lg==",
            "headers":[]
        }
    ]
}
```

Misalkan produser Apache Kafka Anda menulis pesan ke topik Anda dalam format JSON berikut.

```
{
    "device_ID": "AB1234",
```

```
    "session":{
        "start_time": "yyyy-mm-ddThh:mm:ss",
        "duration": 162
    }
}
```

Anda dapat menggunakan `value` kunci untuk memfilter catatan. Misalkan Anda ingin memfilter hanya catatan-catatan di mana `device_ID` dimulai dengan huruf AB. `FilterCriteria`Objeknya adalah sebagai berikut.

```
{
    "Filters": [
        {
            "Pattern": "{ \"value\" : { \"device_ID\" : [ { \"prefix\": \"AB\" } ] } }"
        }
    ]
}
```

Untuk kejelasan tambahan, berikut adalah nilai filter yang `Pattern` diperluas di JSON biasa.

```
{
    "value": {
        "device_ID": [ { "prefix": "AB" } ]
      }
}
```

Anda dapat menambahkan filter menggunakan konsol, AWS CLI atau AWS SAM templat.

Console

Untuk menambahkan filter ini menggunakan konsol, ikuti instruksi Melampirkan kriteria filter ke pemetaan sumber peristiwa (konsol) dan masukkan string berikut untuk kriteria Filter.

```
{ "value" : { "device_ID" : [ { "prefix":  "AB" } ] } }
```

AWS CLI

Untuk membuat pemetaan sumber peristiwa baru dengan kriteria filter ini menggunakan AWS Command Line Interface (AWS CLI), jalankan perintah berikut.

```
aws lambda create-event-source-mapping \
```

```
    --function-name my-function \
    --event-source-arn arn:aws:kafka:us-east-2:123456789012:cluster/my-cluster/
b-8ac7cc01-5898-482d-be2f-a6b596050ea8 \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"value\" : { \"device_ID\" :
 [ { \"prefix\":  \"AB\" } ] } }"}]}'
```

Untuk menambahkan kriteria filter ini ke pemetaan sumber peristiwa yang ada, jalankan perintah berikut.

```
aws lambda update-event-source-mapping \
    --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"value\" : { \"device_ID\" :
 [ { \"prefix\":  \"AB\" } ] } }"}]}'
```

AWS SAM

Untuk menambahkan filter ini menggunakan AWS SAM, tambahkan cuplikan berikut ke template YAMAL untuk sumber acara Anda.

```
FilterCriteria:
  Filters:
    - Pattern: '{ "value" : { "device_ID" : [ { "prefix":  "AB" } ] } }'
```

Dengan Kafka, Anda juga dapat memfilter catatan di mana pesannya adalah string biasa. Misalkan Anda ingin mengabaikan pesan-pesan di mana string adalah “kesalahan”. `FilterCriteria`Objek akan terlihat sebagai berikut.

```
{
    "Filters": [
        {
            "Pattern": "{ \"value\" : [ { \"anything-but\": [ \"error\" ] } ] }"
        }
    ]
}
```

Untuk kejelasan tambahan, berikut adalah nilai filter yang `Pattern` diperluas di JSON biasa.

```
{
    "value": [
        {
```

```
        "anything-but": [ "error" ]
        }
    ]
}
```

Anda dapat menambahkan filter menggunakan konsol, AWS CLI atau AWS SAM templat.

Console

Untuk menambahkan filter ini menggunakan konsol, ikuti instruksi Melampirkan kriteria filter ke pemetaan sumber peristiwa (konsol) dan masukkan string berikut untuk kriteria Filter.

```
{ "value" : [ { "anything-but": [ "error" ] } ] }
```

AWS CLI

Untuk membuat pemetaan sumber peristiwa baru dengan kriteria filter ini menggunakan AWS Command Line Interface (AWS CLI), jalankan perintah berikut.

```
aws lambda create-event-source-mapping \
    --function-name my-function \
    --event-source-arn arn:aws:kafka:us-east-2:123456789012:cluster/my-cluster/
b-8ac7cc01-5898-482d-be2f-a6b596050ea8 \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"value\" : [ { \"anything-but\":
 [ \"error\" ] } ] }"}]}'
```

Untuk menambahkan kriteria filter ini ke pemetaan sumber peristiwa yang ada, jalankan perintah berikut.

```
aws lambda update-event-source-mapping \
    --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"value\" : [ { \"anything-but\":
 [ \"error\" ] } ] }"}]}'
```

AWS SAM

Untuk menambahkan filter ini menggunakan AWS SAM, tambahkan cuplikan berikut ke template YAMAL untuk sumber acara Anda.

```
FilterCriteria:
```

```
  Filters:
    - Pattern: '{ "value" : [ { "anything-but": [ "error" ] } ] }'
```

Pesan Kafka harus berupa string yang UTF-8 dikodekan, baik string biasa atau dalam format JSON. Itu karena Lambda menerjemahkan array byte Kafka ke dalam UTF-8 sebelum menerapkan kriteria filter. Jika pesan Anda menggunakan pengkodean lain, seperti UTF-16 atau ASCII, atau jika format pesan tidak cocok dengan format`FilterCriteria`, Lambda hanya memproses filter metadata. Tabel berikut merangkum perilaku spesifik:

Format pesan masuk	Format pola filter untuk properti pesan	Tindakan yang dihasilkan
Tali polos	Tali polos	Filter Lambda berdasarkan kriteria filter Anda.
Tali polos	Tidak ada pola filter untuk properti data	Filter Lambda (hanya pada properti metadata lainnya) berdasarkan kriteria filter Anda.
Tali polos	JSON yang valid	Filter Lambda (hanya pada properti metadata lainnya) berdasarkan kriteria filter Anda.
JSON yang valid	Tali polos	Filter Lambda (hanya pada properti metadata lainnya) berdasarkan kriteria filter Anda.
JSON yang valid	Tidak ada pola filter untuk properti data	Filter Lambda (hanya pada properti metadata lainnya) berdasarkan kriteria filter Anda.
JSON yang valid	JSON yang valid	Filter Lambda berdasarkan kriteria filter Anda.

Format pesan masuk	Format pola filter untuk properti pesan	Tindakan yang dihasilkan
Non-UTF-8 string yang dikodekan	JSON, string polos, atau tidak ada pola	Filter Lambda (hanya pada properti metadata lainnya) berdasarkan kriteria filter Anda.

Menggunakan pendaftar skema dengan sumber acara Kafka di Lambda

Pendaftaran skema membantu Anda menentukan dan mengelola skema aliran data. Sebuah skema mendefinisikan struktur dan format catatan data. Dalam konteks pemetaan sumber acara Kafka, Anda dapat mengonfigurasi registri skema untuk memvalidasi struktur dan format pesan Kafka terhadap skema yang telah ditentukan sebelum mencapai Fungsi Acara Lambda Anda. Ini menambahkan lapisan tata kelola data ke aplikasi Anda dan Anda dapat mengelola format data secara efisien, memastikan kepatuhan skema, dan mengoptimalkan biaya melalui penyaringan acara.

Fitur ini berfungsi dengan semua bahasa pemrograman, tetapi pertimbangkan poin-poin penting ini:

- Powertools untuk Lambda menyediakan dukungan khusus untuk Java, Python, TypeScript dan, menjaga konsistensi dengan pola pengembangan Kafka yang ada dan memungkinkan akses langsung ke objek bisnis tanpa kode deserialisasi khusus
- Fitur ini hanya tersedia untuk pemetaan sumber peristiwa menggunakan mode yang disediakan. Registri skema tidak mendukung pemetaan sumber peristiwa dalam mode sesuai permintaan. Jika Anda menggunakan mode yang disediakan dan memiliki registri skema yang dikonfigurasi, Anda tidak dapat mengubah ke mode sesuai permintaan kecuali Anda menghapus konfigurasi registri skema terlebih dahulu. Untuk informasi selengkapnya, lihat the section called “Mode yang disediakan”
- Anda hanya dapat mengonfigurasi satu registri skema per pemetaan sumber peristiwa (ESM). Menggunakan registri skema dengan sumber acara Kafka Anda dapat meningkatkan penggunaan Lambda Event Poller Unit (EPU) Anda, yang merupakan dimensi harga untuk mode Provisioned.

Topik

- Opsi registri skema
- Bagaimana Lambda melakukan validasi skema untuk pesan Kafka
- Mengkonfigurasi registri skema Kafka
- Pemfilteran untuk Avro dan Protobuf
- Format muatan dan perilaku deserialisasi
- Bekerja dengan data deserialisasi dalam Fungsi Acara Lambda
- Metode otentikasi untuk registri skema Anda
- Penanganan kesalahan dan pemecahan masalah untuk masalah registri skema

Opsi registri skema

Lambda mendukung opsi registri skema berikut:

- AWS Glue Registri Skema
- Registri Skema Cloud Confluent
- Self-managed Registri Skema Konfluen

Registri skema Anda mendukung memvalidasi pesan dalam format data berikut:

- Apache Avro
- Protokol Buffer (Protobuf)
- Skema JSON () JSON-SE

Untuk menggunakan registri skema, pertama-tama pastikan bahwa pemetaan sumber peristiwa Anda dalam mode yang disediakan. Saat Anda menggunakan registri skema, Lambda menambahkan metadata tentang skema ke payload. Untuk informasi selengkapnya, lihat Format muatan dan perilaku deserialisasi.

Bagaimana Lambda melakukan validasi skema untuk pesan Kafka

Saat Anda mengonfigurasi registri skema, Lambda melakukan langkah-langkah berikut untuk setiap pesan Kafka:

1. Lambda melakukan polling catatan Kafka dari cluster Anda.
2. Lambda memvalidasi atribut pesan yang dipilih dalam catatan terhadap skema tertentu di registri skema Anda.
   - Jika skema yang terkait dengan pesan tidak ditemukan di registri, Lambda mengirim pesan ke DLQ dengan kode alasan. `SCHEMA_NOT_FOUND`
3. Lambda deserialisasi pesan sesuai dengan konfigurasi registri skema untuk memvalidasi pesan. Jika pemfilteran peristiwa dikonfigurasi, Lambda kemudian melakukan pemfilteran berdasarkan kriteria filter yang dikonfigurasi.
   - Jika deserialisasi gagal, Lambda mengirim pesan ke DLQ dengan kode alasan. `DESERIALIZATION_ERROR` Jika tidak ada DLQ yang dikonfigurasi, Lambda akan menghapus pesan.

4. Jika pesan divalidasi oleh registri skema, dan tidak disaring berdasarkan kriteria filter, Lambda akan memanggil fungsi Anda dengan pesan tersebut.

Fitur ini dimaksudkan untuk memvalidasi pesan yang sudah diproduksi menggunakan klien Kafka yang terintegrasi dengan registri skema. Kami merekomendasikan untuk mengonfigurasi produsen Kafka Anda untuk bekerja dengan registri skema Anda untuk membuat pesan yang diformat dengan benar.

Mengkonfigurasi registri skema Kafka

Langkah-langkah konsol berikut menambahkan konfigurasi registri skema Kafka ke pemetaan sumber acara Anda.

Untuk menambahkan konfigurasi registri skema Kafka ke pemetaan sumber acara Anda (konsol)

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih Konfigurasi.
3. Pilih Pemicu.
4. Pilih pemetaan sumber peristiwa Kafka yang ingin Anda konfigurasikan registri skema, dan pilih Edit.
5. Di bawah konfigurasi poller acara, pilih Konfigurasi registri skema. Pemetaan sumber acara Anda harus dalam mode yang disediakan untuk melihat opsi ini.
6. Untuk URI registri Skema, masukkan ARN registri skema AWS Glue Anda, atau URL HTTPS dari registri skema Confluent Cloud atau Registry Skema Konfluen Anda. Self-Managed
7. Langkah-langkah konfigurasi berikut memberi tahu Lambda cara mengakses registri skema Anda. Untuk informasi selengkapnya, lihat ???.
   - Untuk tipe konfigurasi Access, pilih jenis otentikasi yang digunakan Lambda untuk mengakses registri skema Anda.
   - Untuk URI konfigurasi Access, masukkan ARN rahasia Secrets Manager untuk mengautentikasi dengan registri skema Anda, jika berlaku. Pastikan peran eksekusi fungsi Anda berisi izin yang benar.
8. Bidang Enkripsi hanya berlaku jika registri skema Anda ditandatangani oleh Otoritas Sertifikat (CA) pribadi atau otoritas sertifikat (CA) yang tidak ada di toko kepercayaan Lambda.. Jika berlaku, berikan kunci rahasia yang berisi sertifikat CA pribadi yang digunakan oleh registri skema Anda untuk enkripsi TLS.

9. Untuk format rekaman Peristiwa, pilih bagaimana Anda ingin Lambda mengirimkan catatan fungsi Anda setelah validasi skema. Untuk informasi selengkapnya, lihat Contoh format payload.
   - Jika Anda memilih JSON, Lambda memberikan atribut yang Anda pilih dalam atribut validasi Skema di bawah ini dalam format JSON standar. Untuk atribut yang tidak Anda pilih, Lambda mengirimkannya apa adanya.
   - Jika Anda memilih SUMBER, Lambda memberikan atribut yang Anda pilih dalam atribut validasi Skema di bawah ini dalam format sumber aslinya.
10. Untuk atribut validasi Skema, pilih atribut pesan yang Anda ingin Lambda untuk memvalidasi dan deserialisasi menggunakan registri skema Anda. Anda harus memilih setidaknya satu dari KEY atau VALUE. Jika Anda memilih JSON untuk format rekaman peristiwa, Lambda juga melakukan deserialisasi atribut pesan yang dipilih sebelum mengirimnya ke fungsi Anda. Untuk informasi selengkapnya, lihat Format muatan dan perilaku deserialisasi.
11. Pilih Simpan.

Anda juga dapat menggunakan API Lambda untuk membuat atau memperbarui pemetaan sumber peristiwa Anda dengan konfigurasi registri skema. Contoh berikut menunjukkan cara mengonfigurasi registri skema AWS Glue atau Confluent menggunakan AWS CLI, yang sesuai dengan operasi UpdateEventSourceMappingdan CreateEventSourceMappingAPI di Referensi API:AWS Lambda

> **Important**
>
> Jika Anda memperbarui kolom konfigurasi registri skema apa pun menggunakan AWS CLI atau `update-event-source-mapping` API, Anda harus memperbarui semua bidang konfigurasi registri skema.

Create Event Source Mapping

```
aws lambda create-event-source-mapping \
  --function-name my-schema-validator-function \
  --event-source-arn arn:aws:kafka:us-east-1:123456789012:cluster/my-cluster/
a1b2c3d4-5678-90ab-cdef-11111EXAMPLE \
  --topics my-kafka-topic \
  --provisioned-poller-config MinimumPollers=1,MaximumPollers=1 \
  --amazon-managed-kafka-event-source-mapping '{
      "SchemaRegistryConfig" : {
          "SchemaRegistryURI": "https://abcd-ef123.us-west-2.aws.confluent.cloud",
```

```
          "AccessConfigs": [{
              "Type": "BASIC_AUTH",
              "URI": "arn:aws:secretsmanager:us-
east-1:123456789012:secret:secretName"
          }],
          "EventRecordFormat": "JSON",
          "SchemaValidationConfigs": [
          {
              "Attribute": "KEY"
          },
          {
              "Attribute": "VALUE"
          }]
      }
  }'
```

Update AWS Glue Schema Registry

```
aws lambda update-event-source-mapping \
    --uuid a1b2c3d4-5678-90ab-cdef-EXAMPLE11111 \
    --amazon-managed-kafka-event-source-mapping '{
        "SchemaRegistryConfig" : {
            "SchemaRegistryURI": "arn:aws:glue:us-east-1:123456789012:registry/
registryName",
            "EventRecordFormat": "JSON",
            "SchemaValidationConfigs": [
            {
                "Attribute": "KEY"
            },
            {
                "Attribute": "VALUE"
            }]
        }
    }'
```

Update Confluent Schema Registry with Authentication

```
aws lambda update-event-source-mapping \
    --uuid a1b2c3d4-5678-90ab-cdef-EXAMPLE11111 \
    --amazon-managed-kafka-event-source-mapping '{
        "SchemaRegistryConfig" : {
            "SchemaRegistryURI": "https://abcd-ef123.us-west-2.aws.confluent.cloud",
            "AccessConfigs": [{
```

```
                "Type": "BASIC_AUTH",
                "URI": "arn:aws:secretsmanager:us-
east-1:123456789012:secret:secretName"
            }],
            "EventRecordFormat": "JSON",
            "SchemaValidationConfigs": [
            {
                "Attribute": "KEY"
            },
            {
                "Attribute": "VALUE"
            }]
        }
    }'
```

Update Confluent Schema Registry without Authentication

```
aws lambda update-event-source-mapping \
    --uuid a1b2c3d4-5678-90ab-cdef-EXAMPLE11111 \
    --amazon-managed-kafka-event-source-mapping '{
        "SchemaRegistryConfig" : {
            "SchemaRegistryURI": "https://abcd-ef123.us-west-2.aws.confluent.cloud",
            "EventRecordFormat": "JSON",
            "SchemaValidationConfigs": [
            {
                "Attribute": "KEY"
            },
            {
                "Attribute": "VALUE"
            }]
        }
    }'
```

Remove Schema Registry Configuration

Untuk menghapus konfigurasi registri skema dari pemetaan sumber peristiwa, Anda dapat menggunakan perintah CLI UpdateEventSourceMappingdi Referensi API.AWS Lambda

```
aws lambda update-event-source-mapping \
    --uuid a1b2c3d4-5678-90ab-cdef-EXAMPLE11111 \
    --amazon-managed-kafka-event-source-mapping '{
        "SchemaRegistryConfig" : {}
    }'
```

Pemfilteran untuk Avro dan Protobuf

Saat menggunakan format Avro atau Protobuf dengan registri skema, Anda dapat menerapkan pemfilteran acara ke Fungsi Acara Lambda Anda. Pola filter diterapkan pada representasi JSON klasik deserialisasi dari data Anda setelah validasi skema. Misalnya, dengan skema Avro yang menentukan detail produk termasuk harga, Anda dapat memfilter pesan berdasarkan nilai harga:

> **Note**
>
> Saat dideserialisasi, Avro dikonversi ke JSON standar, yang berarti tidak dapat langsung dikonversi kembali ke objek Avro. Jika Anda perlu mengonversi ke objek Avro, gunakan format SOURCE sebagai gantinya.
> Untuk deserialisasi Protobuf, nama bidang di JSON yang dihasilkan cocok dengan yang ditentukan dalam skema Anda, daripada dikonversi ke kasus unta seperti yang biasanya dilakukan Protobuf. Ingatlah hal ini saat membuat pola penyaringan.

```
aws lambda create-event-source-mapping \
    --function-name myAvroFunction \
    --topics myAvroTopic \
    --starting-position TRIM_HORIZON \
    --kafka-bootstrap-servers '["broker1:9092", "broker2:9092"]' \
    --schema-registry-config '{
        "SchemaRegistryURI": "arn:aws:glue:us-east-1:123456789012:registry/
myAvroRegistry",
        "EventRecordFormat": "JSON",
        "SchemaValidationConfigs": [
            {
                "Attribute": "VALUE"
            }
        ]
    }' \
    --filter-criteria '{
        "Filters": [
            {
                "Pattern": "{ \"value\" : { \"field_1\" : [\"value1\"], \"field_2\" :
 [\"value2\"] } }"
            }
        ]
    }'
```

Dalam contoh ini, pola filter menganalisis `value` objek, mencocokkan pesan dengan `"value1"` dan `field_1 field_2` dengan`"value2"`. Kriteria filter dievaluasi terhadap data deserialisasi, setelah Lambda mengonversi pesan dari format Avro ke JSON.

Untuk informasi lebih rinci tentang pemfilteran acara, lihat Pemfilteran acara Lambda.

Format muatan dan perilaku deserialisasi

Saat menggunakan registri skema, Lambda mengirimkan muatan akhir ke fungsi Anda dalam format yang mirip dengan muatan acara biasa, dengan beberapa bidang tambahan. Bidang tambahan tergantung pada `SchemaValidationConfigs` parameter. Untuk setiap atribut yang Anda pilih untuk validasi (kunci atau nilai), Lambda menambahkan metadata skema yang sesuai ke payload.

> **Note**
>
> Anda harus memperbarui aws-lambda-java-events ke versi 3.16.0 atau lebih tinggi untuk menggunakan bidang metadata skema.

Misalnya, jika Anda memvalidasi `value` bidang, Lambda menambahkan bidang yang `valueSchemaMetadata` dipanggil ke payload Anda. Demikian pula, untuk `key` bidang, Lambda menambahkan bidang yang disebut. `keySchemaMetadata` Metadata ini berisi informasi tentang format data dan ID skema yang digunakan untuk validasi:

```
"valueSchemaMetadata": {
    "dataFormat": "AVRO",
    "schemaId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
}
```

`EventRecordFormat`Parameter dapat diatur ke salah satu `JSON` atau`SOURCE`, yang menentukan cara Lambda menangani data yang divalidasi skema sebelum mengirimkannya ke fungsi Anda. Setiap opsi menyediakan kemampuan pemrosesan yang berbeda:

- `JSON`- Lambda deserialisasi atribut yang divalidasi ke dalam format JSON standar, membuat data siap untuk digunakan langsung dalam bahasa dengan dukungan JSON asli. Format ini sangat ideal ketika Anda tidak perlu mempertahankan format biner asli atau bekerja dengan kelas yang dihasilkan.
- `SOURCE`- Lambda mempertahankan format biner asli dari data sebagai Base64-encoded string, memungkinkan konversi langsung ke objek Avro atau Protobuf. Format ini sangat penting saat

bekerja dengan bahasa yang diketik dengan kuat atau ketika Anda perlu mempertahankan kemampuan penuh skema Avro atau Protobuf.

Berdasarkan karakteristik format dan pertimbangan khusus bahasa ini, kami merekomendasikan format berikut:

Format yang disarankan berdasarkan bahasa pemrograman

Bahasa	Avro	Protobuf	JSON
Java	SUMBER	SUMBER	SUMBER
Python	JSON	JSON	JSON
NodeJS	JSON	JSON	JSON
.NET	SUMBER	SUMBER	SUMBER
Lainnya	JSON	JSON	JSON

Bagian berikut menjelaskan format ini secara rinci dan memberikan contoh muatan untuk setiap format.

Format JSON

Jika Anda memilih `JSON` sebagai`EventRecordFormat`, Lambda memvalidasi dan deserialisasi atribut pesan yang telah Anda pilih di `SchemaValidationConfigs` bidang (atribut). `key` and/or `value` Lambda memberikan atribut yang dipilih ini sebagai string yang dikodekan base64 dari representasi JSON standar mereka dalam fungsi Anda.

> **Note**
>
> Saat dideserialisasi, Avro dikonversi ke JSON standar, yang berarti tidak dapat langsung dikonversi kembali ke objek Avro. Jika Anda perlu mengonversi ke objek Avro, gunakan format SOURCE sebagai gantinya.
> Untuk deserialisasi Protobuf, nama bidang di JSON yang dihasilkan cocok dengan yang ditentukan dalam skema Anda, daripada dikonversi ke kasus unta seperti yang biasanya dilakukan Protobuf. Ingatlah hal ini saat membuat pola penyaringan.

Berikut ini menunjukkan contoh payload, dengan asumsi Anda memilih JSON sebagai`EventRecordFormat`, dan kedua `value` atribut `key` dan sebagai: `SchemaValidationConfigs`

```
{
   "eventSource":"aws:kafka",
   "eventSourceArn":"arn:aws:kafka:us-east-1:123456789012:cluster/vpc-2priv-2pub/
a1b2c3d4-5678-90ab-cdef-EXAMPLE11111-1",
   "bootstrapServers":"b-2.demo-cluster-1.a1bcde.c1.kafka.us-
east-1.amazonaws.com:9092,b-1.demo-cluster-1.a1bcde.c1.kafka.us-
east-1.amazonaws.com:9092",
   "records":{
      "mytopic-0":[
         {
            "topic":"mytopic",
            "partition":0,
            "offset":15,
            "timestamp":1545084650987,
            "timestampType":"CREATE_TIME",
            "key":"abcDEFghiJKLmnoPQRstuVWXyz1234==", //Base64 encoded string of JSON
            "keySchemaMetadata": {
                "dataFormat": "AVRO",
                "schemaId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
            },
            "value":"abcDEFghiJKLmnoPQRstuVWXyz1234", //Base64 encoded string of JSON
            "valueSchemaMetadata": {
                "dataFormat": "AVRO",
                "schemaId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
            },
            "headers":[
               {
                  "headerKey":[
                     104,
                     101,
                     97,
                     100,
                     101,
                     114,
                     86,
                     97,
                     108,
                     117,
                     101
```

```
                    ]
                }
            ]
        }
    ]
  }
}
```

Dalam contoh ini:

- Keduanya `key` dan `value` merupakan string yang dikodekan base64 dari representasi JSON mereka setelah deserialisasi.
- Lambda menyertakan metadata skema untuk kedua atribut di dan. `keySchemaMetadata` `valueSchemaMetadata`
- Fungsi Anda dapat memecahkan kode `key` dan `value` string untuk mengakses data JSON deserialized.

Format JSON direkomendasikan untuk bahasa yang tidak diketik dengan kuat, seperti Python atau. Node.js Bahasa-bahasa ini memiliki dukungan asli untuk mengubah JSON menjadi objek.

Format sumber

Jika Anda memilih `SOURCE` sebagai`EventRecordFormat`, Lambda masih memvalidasi catatan terhadap registri skema, tetapi mengirimkan data biner asli ke fungsi Anda tanpa deserialisasi. Data biner ini dikirimkan sebagai string yang dikodekan Base64 dari data byte asli, dengan metadata yang ditambahkan produsen dihapus. Akibatnya, Anda dapat langsung mengonversi data biner mentah menjadi objek Avro dan Protobuf dalam kode fungsi Anda. Kami merekomendasikan menggunakan Powertools untuk AWS Lambda, yang deserialisasi data biner mentah dan memberi Anda objek Avro dan Protobuf secara langsung.

Misalnya, jika Anda mengonfigurasi Lambda untuk memvalidasi `value` atribut `key` dan atribut namun menggunakan `SOURCE` formatnya, fungsi Anda akan menerima payload seperti ini:

```
{
    "eventSource": "aws:kafka",
    "eventSourceArn": "arn:aws:kafka:us-east-1:123456789012:cluster/vpc-2priv-2pub/
a1b2c3d4-5678-90ab-cdef-EXAMPLE11111-1",
    "bootstrapServers": "b-2.demo-cluster-1.a1bcde.c1.kafka.us-
east-1.amazonaws.com:9092,b-1.demo-cluster-1.a1bcde.c1.kafka.us-
east-1.amazonaws.com:9092",
```

```
    "records": {
        "mytopic-0": [
            {
                "topic": "mytopic",
                "partition": 0,
                "offset": 15,
                "timestamp": 1545084650987,
                "timestampType": "CREATE_TIME",
                "key": "abcDEFghiJKLmnoPQRstuVWXyz1234==", // Base64 encoded string of
 Original byte data, producer-appended metadata removed
                "keySchemaMetadata": {
                    "dataFormat": "AVRO",
                    "schemaId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
                },
                "value": "abcDEFghiJKLmnoPQRstuVWXyz1234==", // Base64 encoded string
 of Original byte data, producer-appended metadata removed
                "valueSchemaMetadata": {
                    "dataFormat": "AVRO",
                    "schemaId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
                },
                "headers": [
                    {
                        "headerKey": [
                            104,
                            101,
                            97,
                            100,
                            101,
                            114,
                            86,
                            97,
                            108,
                            117,
                            101
                        ]
                    }
                ]
            }
        ]
    }
}
```

Dalam contoh ini:

- Keduanya `key` dan `value` berisi data biner asli sebagai string yang dikodekan Base64.
- Fungsi Anda perlu menangani deserialisasi menggunakan pustaka yang sesuai.

Memilih `SOURCE` untuk `EventRecordFormat` disarankan jika Anda menggunakan Avro-generated atau Protobuf-generated objek, terutama dengan fungsi Java. Ini karena Java diketik dengan kuat, dan membutuhkan deserializer khusus untuk format Avro dan Protobuf. Dalam kode fungsi Anda, Anda dapat menggunakan pustaka Avro atau Protobuf pilihan Anda untuk deserialisasi data.

Bekerja dengan data deserialisasi dalam Fungsi Acara Lambda

Powertools untuk AWS Lambda membantu Anda deserialisasi catatan Kafka dalam kode fungsi Anda berdasarkan format yang Anda gunakan. Utilitas ini menyederhanakan bekerja dengan catatan Kafka dengan menangani konversi data dan menyediakan objek siap pakai.

Untuk menggunakan Powertools untuk AWS Lambda fungsi Anda, Anda perlu menambahkan Powertools AWS Lambda baik sebagai layer atau memasukkannya sebagai dependensi saat membangun Lambda Event Function Anda. Untuk petunjuk penyiapan dan informasi selengkapnya, lihat Powertools untuk AWS Lambda dokumentasi untuk bahasa pilihan Anda:

- Powertools untuk AWS Lambda (Java)
- Powertools untuk AWS Lambda (Python)
- Powertools untuk AWS Lambda () TypeScript
- Powertools untuk AWS Lambda (.NET)

> **Note**
>
> Saat bekerja dengan integrasi registri skema, Anda dapat memilih `SOURCE` atau `JSON` memformat. Setiap opsi mendukung format serialisasi yang berbeda seperti yang ditunjukkan di bawah ini:

Format	Mendukung
SUMBER	Avro dan Protobuf (menggunakan integrasi Lambda Schema Registry)
JSON	Data JSON

Saat menggunakan `JSON` format `SOURCE` or, Anda dapat menggunakan Powertools AWS untuk membantu deserialisasi data dalam kode fungsi Anda. Berikut adalah contoh cara menangani format data yang berbeda:

AVRO

Contoh Java:

```
package org.demo.kafka;

import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.demo.kafka.avro.AvroProduct;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;

import software.amazon.lambda.powertools.kafka.Deserialization;
import software.amazon.lambda.powertools.kafka.DeserializationType;
import software.amazon.lambda.powertools.logging.Logging;

public class AvroDeserializationFunction implements
 RequestHandler<ConsumerRecords<String, AvroProduct>, String> {

    private static final Logger LOGGER =
 LoggerFactory.getLogger(AvroDeserializationFunction.class);

    @Override
    @Logging
    @Deserialization(type = DeserializationType.KAFKA_AVRO)
    public String handleRequest(ConsumerRecords<String, AvroProduct> records,
 Context context) {
        for (ConsumerRecord<String, AvroProduct> consumerRecord : records) {
            LOGGER.info("ConsumerRecord: {}", consumerRecord);

            AvroProduct product = consumerRecord.value();
            LOGGER.info("AvroProduct: {}", product);

            String key = consumerRecord.key();
            LOGGER.info("Key: {}", key);
        }
```

```
        return "OK";
    }

}
```

Contoh Python:

```
from aws_lambda_powertools.utilities.kafka_consumer.kafka_consumer import
 kafka_consumer
from aws_lambda_powertools.utilities.kafka_consumer.schema_config import
 SchemaConfig
from aws_lambda_powertools.utilities.kafka_consumer.consumer_records import
 ConsumerRecords

from aws_lambda_powertools.utilities.typing import LambdaContext
from aws_lambda_powertools import Logger

logger = Logger(service="kafkaConsumerPowertools")

value_schema_str = open("customer_profile.avsc", "r").read()

schema_config = SchemaConfig(
value_schema_type="AVRO",
value_schema=value_schema_str)

@kafka_consumer(schema_config=schema_config)
def lambda_handler(event: ConsumerRecords, context:LambdaContext):

  for record in event.records:
      value = record.value
      logger.info(f"Received value: {value}")
```

TypeScript contoh:

```
import { kafkaConsumer } from '@aws-lambda-powertools/kafka';

import type { ConsumerRecords } from '@aws-lambda-powertools/kafka/types';
import { Logger } from '@aws-lambda-powertools/logger';
import type { Context } from 'aws-lambda';

const logger = new Logger();
```

```
type Value = {
    id: number;
    name: string;
    price: number;
};

const schema = '{
    "type": "record",
    "name": "Product",
    "fields": [
        { "name": "id", "type": "int" },
        { "name": "name", "type": "string" },
        { "name": "price", "type": "double" }
    ]
}';

export const handler = kafkaConsumer<string, Value>(
    (event: ConsumerRecords<string, Value>, _context: Context) => {
        for (const record of event.records) {
            logger.info(Processing record with key: ${record.key});
            logger.info(Record value: ${JSON.stringify(record.value)});
            // You can add more processing logic here
        }
    },
    {
        value: {
            type: 'avro',
            schema: schema,
        },
    }
);
```

Contoh .NET:

```
using Amazon.Lambda.Core;
using AWS.Lambda.Powertools.Kafka;
using AWS.Lambda.Powertools.Kafka.Avro;
using AWS.Lambda.Powertools.Logging;
using Com.Example;

// Assembly attribute to enable the Lambda Event Function's Kafka event to be
 converted into a .NET class.
[assembly: LambdaSerializer(typeof(PowertoolsKafkaAvroSerializer))]
```

```
namespace ProtoBufClassLibrary;

public class Function
{
    public string FunctionHandler(ConsumerRecords<string, CustomerProfile> records,
 ILambdaContext context)
    {
        foreach (var record in records)
        {
            Logger.LogInformation("Processing messagem from topic: {topic}",
 record.Topic);
            Logger.LogInformation("Partition: {partition}, Offset: {offset}",
 record.Partition, record.Offset);
            Logger.LogInformation("Produced at: {timestamp}", record.Timestamp);

            foreach (var header in record.Headers.DecodedValues())
            {
                Logger.LogInformation($"{header.Key}: {header.Value}");
            }

            Logger.LogInformation("Processing order for: {fullName}",
 record.Value.FullName);
        }

        return "Processed " + records.Count() + " records";
    }
}
```

PROTOBUF

Contoh Java:

```
package org.demo.kafka;

import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.demo.kafka.protobuf.ProtobufProduct;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
```

```
import software.amazon.lambda.powertools.kafka.Deserialization;
import software.amazon.lambda.powertools.kafka.DeserializationType;
import software.amazon.lambda.powertools.logging.Logging;

public class ProtobufDeserializationFunction
        implements RequestHandler<ConsumerRecords<String, ProtobufProduct>, String>
 {

    private static final Logger LOGGER =
 LoggerFactory.getLogger(ProtobufDeserializationFunction.class);

    @Override
    @Logging
    @Deserialization(type = DeserializationType.KAFKA_PROTOBUF)
    public String handleRequest(ConsumerRecords<String, ProtobufProduct> records,
 Context context) {
        for (ConsumerRecord<String, ProtobufProduct> consumerRecord : records) {
            LOGGER.info("ConsumerRecord: {}", consumerRecord);

            ProtobufProduct product = consumerRecord.value();
            LOGGER.info("ProtobufProduct: {}", product);

            String key = consumerRecord.key();
            LOGGER.info("Key: {}", key);
        }

        return "OK";
    }

}
```

Contoh Python:

```
from aws_lambda_powertools.utilities.kafka_consumer.kafka_consumer import
 kafka_consumer

from aws_lambda_powertools.utilities.kafka_consumer.schema_config import
 SchemaConfig
from aws_lambda_powertools.utilities.kafka_consumer.consumer_records import
 ConsumerRecords

from aws_lambda_powertools.utilities.typing import LambdaContext
from aws_lambda_powertools import Logger
```

```
from user_pb2 import User # protobuf generated class

logger = Logger(service="kafkaConsumerPowertools")

schema_config = SchemaConfig(
value_schema_type="PROTOBUF",
value_schema=User)

@kafka_consumer(schema_config=schema_config)
def lambda_handler(event: ConsumerRecords, context:LambdaContext):

  for record in event.records:
      value = record.value
      logger.info(f"Received value: {value}")
```

TypeScript contoh:

```
import { kafkaConsumer } from '@aws-lambda-powertools/kafka';
import type { ConsumerRecords } from '@aws-lambda-powertools/kafka/types';
import { Logger } from '@aws-lambda-powertools/logger';
import type { Context } from 'aws-lambda';
import { Product } from './product.generated.js';

const logger = new Logger();

type Value = {
    id: number;
    name: string;
    price: number;
};

export const handler = kafkaConsumer<string, Value>(
    (event: ConsumerRecords<string, Value>, _context: Context) => {
        for (const record of event.records) {
            logger.info(Processing record with key: ${record.key});
            logger.info(Record value: ${JSON.stringify(record.value)});
        }
    },
    {
        value: {
            type: 'protobuf',
            schema: Product,
```

```
        },
    }
);
```

Contoh .NET:

```
using Amazon.Lambda.Core;
using AWS.Lambda.Powertools.Kafka;
using AWS.Lambda.Powertools.Kafka.Protobuf;
using AWS.Lambda.Powertools.Logging;
using Com.Example;

// Assembly attribute to enable the Lambda function's Kafka event to be converted
 into a .NET class.
[assembly: LambdaSerializer(typeof(PowertoolsKafkaProtobufSerializer))]

namespace ProtoBufClassLibrary;

public class Function
{
    public string FunctionHandler(ConsumerRecords<string, CustomerProfile> records,
 ILambdaContext context)
    {
        foreach (var record in records)
        {
            Logger.LogInformation("Processing messagem from topic: {topic}",
 record.Topic);
            Logger.LogInformation("Partition: {partition}, Offset: {offset}",
 record.Partition, record.Offset);
            Logger.LogInformation("Produced at: {timestamp}", record.Timestamp);

            foreach (var header in record.Headers.DecodedValues())
            {
                Logger.LogInformation($"{header.Key}: {header.Value}");
            }

            Logger.LogInformation("Processing order for: {fullName}",
 record.Value.FullName);
        }

        return "Processed " + records.Count() + " records";
    }
}
```

JSON

Contoh Java:

```
package org.demo.kafka;

import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;

import software.amazon.lambda.powertools.kafka.Deserialization;
import software.amazon.lambda.powertools.kafka.DeserializationType;
import software.amazon.lambda.powertools.logging.Logging;

public class JsonDeserializationFunction implements
 RequestHandler<ConsumerRecords<String, Product>, String> {

    private static final Logger LOGGER =
 LoggerFactory.getLogger(JsonDeserializationFunction.class);

    @Override
    @Logging
    @Deserialization(type = DeserializationType.KAFKA_JSON)
    public String handleRequest(ConsumerRecords<String, Product> consumerRecords,
 Context context) {
        for (ConsumerRecord<String, Product> consumerRecord : consumerRecords) {
            LOGGER.info("ConsumerRecord: {}", consumerRecord);

            Product product = consumerRecord.value();
            LOGGER.info("Product: {}", product);

            String key = consumerRecord.key();
            LOGGER.info("Key: {}", key);
        }

        return "OK";
    }
}
```

Contoh Python:

```
from aws_lambda_powertools.utilities.kafka_consumer.kafka_consumer import
 kafka_consumer

from aws_lambda_powertools.utilities.kafka_consumer.schema_config import
 SchemaConfig
from aws_lambda_powertools.utilities.kafka_consumer.consumer_records import
 ConsumerRecords

from aws_lambda_powertools.utilities.typing import LambdaContext
from aws_lambda_powertools import Logger

logger = Logger(service="kafkaConsumerPowertools")

schema_config = SchemaConfig(value_schema_type="JSON")

@kafka_consumer(schema_config=schema_config)
def lambda_handler(event: ConsumerRecords, context:LambdaContext):

  for record in event.records:
      value = record.value
      logger.info(f"Received value: {value}")
```

TypeScript contoh:

```
import { kafkaConsumer } from '@aws-lambda-powertools/kafka';
import type { ConsumerRecords } from '@aws-lambda-powertools/kafka/types';
import { Logger } from '@aws-lambda-powertools/logger';
import type { Context } from 'aws-lambda';

const logger = new Logger();

type Value = {
    id: number;
    name: string;
    price: number;
};

export const handler = kafkaConsumer<string, Value>(
    (event: ConsumerRecords<string, Value>, _context: Context) => {
        for (const record of event.records) {
            logger.info(Processing record with key: ${record.key});
```

```
            logger.info(Record value: ${JSON.stringify(record.value)});
            // You can add more processing logic here
        }
    },
    {
        value: {
            type: 'json',
        },
    }
);
```

Contoh .NET:

```
using Amazon.Lambda.Core;
using AWS.Lambda.Powertools.Kafka;
using AWS.Lambda.Powertools.Kafka.Json;
using AWS.Lambda.Powertools.Logging;
using Com.Example;

// Assembly attribute to enable the Lambda function's Kafka event to be converted
 into a .NET class.
[assembly: LambdaSerializer(typeof(PowertoolsKafkaJsonSerializer))]

namespace JsonClassLibrary;

public class Function
{
    public string FunctionHandler(ConsumerRecords<string, CustomerProfile> records,
 ILambdaContext context)
    {
        foreach (var record in records)
        {
            Logger.LogInformation("Processing messagem from topic: {topic}",
 record.Topic);
            Logger.LogInformation("Partition: {partition}, Offset: {offset}",
 record.Partition, record.Offset);
            Logger.LogInformation("Produced at: {timestamp}", record.Timestamp);

            foreach (var header in record.Headers.DecodedValues())
            {
                Logger.LogInformation($"{header.Key}: {header.Value}");
            }
```

```
            Logger.LogInformation("Processing order for: {fullName}",
 record.Value.FullName);
        }

        return "Processed " + records.Count() + " records";
    }
}
```

Metode otentikasi untuk registri skema Anda

Untuk menggunakan registri skema, Lambda harus dapat mengaksesnya dengan aman. Jika Anda bekerja dengan registri AWS Glue skema, Lambda bergantung pada otentikasi IAM. Ini berarti bahwa peran eksekusi fungsi Anda harus memiliki izin berikut untuk mengakses AWS Glue registri:

- GetRegistrydi Referensi API AWS Glue Web
- GetSchemaVersiondi Referensi API AWS Glue Web

Contoh kebijakan IAM yang diperlukan:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:GetRegistry",
                "glue:GetSchemaVersion"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

> **Note**
>
> Untuk pendaftar AWS Glue skema, jika Anda menyediakan `AccessConfigs` registri AWS Glue , Lambda mengembalikan pengecualian validasi.

Jika Anda bekerja dengan registri skema Confluent, Anda dapat memilih salah satu dari empat metode otentikasi yang didukung untuk `Type` parameter objek Anda: KafkaSchemaRegistryAccessConfig

- BASIC_AUTH — Lambda menggunakan nama pengguna dan kata sandi atau otentikasi Kunci API dan Rahasia API untuk mengakses registri Anda. Jika Anda memilih opsi ini, berikan Secrets Manager ARN yang berisi kredensi Anda di bidang URI.
- CLIENT_CERTIFICATE_TLS_AUTH — Lambda menggunakan otentikasi TLS timbal balik dengan sertifikat klien. Untuk menggunakan opsi ini, Lambda memerlukan akses ke sertifikat dan kunci pribadi. Berikan Secrets Manager ARN yang berisi kredensi ini di bidang URI.
- OAUTHBEARER_AUTH — Lambda meminta token akses dari penyedia identitas OAuth 2.0 Anda dan mengirimkannya ke registri Anda sebagai token pembawa. Lambda menyegarkan token sebelum kedaluwarsa. Jika Anda memilih opsi ini, berikan Secrets Manager ARN yang berisi rahasia OAuth Anda di bidang URI. Rahasia menggunakan bidang yang sama dengan rahasia untuk otentikasi broker, dan mendukung dua jenis hibah yang sama. Untuk isi rahasianya, lihatthe section called “Mengkonfigurasi rahasia OAuth”.
- NO_AUTH - Sertifikat CA publik harus ditandatangani oleh otoritas sertifikat (CA) yang ada di toko kepercayaan Lambda. Untuk CA/self-signed sertifikat pribadi, Anda mengonfigurasi sertifikat CA root server. Untuk menggunakan opsi ini, hilangkan `AccessConfigs` parameter.

Selain itu, jika Lambda memerlukan akses ke sertifikat CA pribadi untuk memverifikasi sertifikat TLS registri skema Anda, pilih `SERVER_ROOT_CA_CERT` sebagai `Type` dan berikan Secrets Manager ARN ke sertifikat di bidang URI.

> **Note**
>
> Untuk mengkonfigurasi `SERVER_ROOT_CA_CERT` opsi di konsol, berikan ARN rahasia yang berisi sertifikat di bidang Enkripsi.

Konfigurasi otentikasi untuk registri skema Anda terpisah dari otentikasi apa pun yang telah Anda konfigurasi untuk cluster Kafka Anda. Anda harus mengkonfigurasi keduanya secara terpisah, bahkan jika mereka menggunakan metode otentikasi serupa.

Penanganan kesalahan dan pemecahan masalah untuk masalah registri skema

Saat menggunakan registri skema dengan sumber acara MSK Amazon Anda, Anda mungkin mengalami berbagai kesalahan. Bagian ini memberikan panduan tentang masalah umum dan cara mengatasinya.

Kesalahan konfigurasi

Kesalahan ini terjadi saat mengatur konfigurasi registri skema Anda.

Diperlukan mode yang disediakan

- Pesan kesalahan: `SchemaRegistryConfig is only available for Provisioned Mode. To configure Schema Registry, please enable Provisioned Mode by specifying MinimumPollers in ProvisionedPollerConfig.`
- Resolusi: Aktifkan mode yang disediakan untuk pemetaan sumber acara Anda dengan mengonfigurasi parameter di. `MinimumPollers` `ProvisionedPollerConfig`

URL registri skema tidak valid

- Pesan kesalahan: `Malformed SchemaRegistryURI provided. Please provide a valid URI or ARN. For example, https://schema-registry.example.com:8081 or arn:aws:glue:us-east-1:123456789012:registry/ExampleRegistry.`
- Resolusi: Berikan URL HTTPS yang valid untuk Confluent Schema Registry atau ARN yang valid untuk Schema Registry. AWS Glue

Format rekaman acara tidak valid atau hilang

- Pesan kesalahan: `EventRecordFormat is a required field for SchemaRegistryConfig. Please provide one of supported format types: SOURCE, JSON.`
- Resolusi: Tentukan SOURCE atau JSON sebagai EventRecordFormat konfigurasi registri skema Anda.

Atribut validasi duplikat

Pesan kesalahan: `Duplicate KEY/VALUE Attribute in SchemaValidationConfigs. SchemaValidationConfigs must contain at most one KEY/VALUE Attribute.`

Resolusi: Hapus duplikat KEY atau atribut VALUE dari atribut Anda SchemaValidationConfigs. Setiap jenis atribut hanya dapat muncul sekali.

Konfigurasi validasi tidak ada

Pesan kesalahan: `SchemaValidationConfigs is a required field for SchemaRegistryConfig.`

Resolusi: Tambahkan SchemaValidationConfigs ke konfigurasi Anda, tentukan setidaknya satu atribut validasi (KEY atau VALUE).

Kesalahan akses dan izin

Kesalahan ini terjadi ketika Lambda tidak dapat mengakses registri skema karena masalah izin atau otentikasi.

AWS Glue Akses registri skema ditolak

Pesan kesalahan: `Cannot access Glue Schema with provided role. Please ensure the provided role can perform the GetRegistry and GetSchemaVersion Actions on your schema.`

Resolusi: Tambahkan izin yang diperlukan (`glue:GetRegistry`dan`glue:GetSchemaVersion`) ke peran eksekusi fungsi Anda.

Akses Registri Skema Konfluen ditolak

Pesan kesalahan: `Cannot access Confluent Schema with the provided access configuration.`

Resolusi: Verifikasi bahwa kredensyal otentikasi Anda (disimpan di Secrets Manager) sudah benar dan memiliki izin yang diperlukan untuk mengakses registri skema.

Cross-account AWS Glue Registri Skema

Pesan kesalahan: `Cross-account Glue Schema Registry ARN not supported.`

Resolusi: Gunakan Registri AWS Glue Skema yang ada di AWS akun yang sama dengan Fungsi Acara Lambda Anda.

Cross-region AWS Glue Registri Skema

Pesan kesalahan: `Cross-region Glue Schema Registry ARN not supported.`

Resolusi: Gunakan Registri AWS Glue Skema yang berada di wilayah yang sama dengan Fungsi Acara Lambda Anda.

Masalah akses rahasia

Pesan kesalahan: `Lambda received InvalidRequestException from Secrets Manager.`

Resolusi: Verifikasi bahwa peran eksekusi fungsi Anda memiliki izin untuk mengakses rahasia dan bahwa rahasia tidak dienkripsi dengan AWS KMS kunci default jika mengakses dari akun yang berbeda.

Kesalahan koneksi

Kesalahan ini terjadi ketika Lambda tidak dapat membuat koneksi ke registri skema.

Masalah konektivitas VPC

Pesan kesalahan: `Cannot connect to your Schema Registry. Your Kafka cluster's VPC must be able to connect to the schema registry. You can provide access by configuring AWS PrivateLink or a NAT Gateway or VPC Peering between Kafka Cluster VPC and the schema registry VPC.`

Resolusi: Konfigurasikan jaringan VPC Anda untuk memungkinkan koneksi ke registri skema menggunakan AWS PrivateLink, NAT Gateway, atau VPC peering.

Kegagalan jabat tangan TLS

Pesan kesalahan: `Unable to establish TLS handshake with the schema registry. Please provide correct CA-certificate or client certificate using Secrets Manager to access your schema registry.`

Resolusi: Verifikasi bahwa sertifikat CA dan sertifikat klien Anda (untuk mTL) sudah benar dan dikonfigurasi dengan benar di Secrets Manager.

Throttling

Pesan kesalahan: `Receiving throttling errors when accessing the schema registry. Please increase API TPS limits for your schema registry.`

Resolusi: Tingkatkan batas tarif API untuk registri skema Anda atau kurangi laju permintaan dari aplikasi Anda.

Self-managed kesalahan registri skema

Pesan kesalahan: `Lambda received an internal server an unexpected error from the provided self-managed schema registry.`

Resolusi: Periksa kesehatan dan konfigurasi server registri skema yang dikelola sendiri.

Pemrosesan latensi rendah untuk sumber acara Kafka

AWS Lambda secara native mendukung pemrosesan peristiwa latensi rendah untuk aplikasi yang membutuhkan latensi ujung ke ujung yang konsisten kurang dari 100 milidetik. Halaman ini menyediakan detail konfigurasi dan rekomendasi untuk mengaktifkan alur kerja latensi rendah.

Aktifkan pemrosesan latensi rendah

Untuk mengaktifkan pemrosesan latensi rendah pada pemetaan sumber peristiwa Kafka, konfigurasi dasar berikut diperlukan:

- Aktifkan mode yang disediakan. Untuk informasi selengkapnya, lihat Mode yang disediakan.
- Setel `MaximumBatchingWindowInSeconds` parameter pemetaan sumber peristiwa ke 0. Untuk informasi selengkapnya, lihat Perilaku batching.

Fine-tuning Kafka ESM latensi rendah Anda

Pertimbangkan rekomendasi berikut untuk mengoptimalkan pemetaan sumber acara Kafka Anda untuk latensi rendah:

Konfigurasi mode yang disediakan

Dalam mode yang disediakan untuk pemetaan sumber peristiwa Kafka, Anda dapat menyempurnakan throughput pemetaan sumber peristiwa Anda dengan mengonfigurasi jumlah sumber daya minimum dan maksimum yang disebut poller peristiwa. Poller peristiwa (atau poller) mewakili sumber daya komputasi yang mendukung pemetaan sumber peristiwa dalam mode yang disediakan, dan mengalokasikan hingga 5 throughput. MB/s Setiap poller acara mendukung hingga 5 pemanggilan Lambda bersamaan.

Untuk menentukan konfigurasi poller yang optimal untuk aplikasi Anda, pertimbangkan tingkat konsumsi puncak dan persyaratan pemrosesan Anda. Mari kita lihat contoh yang disederhanakan:

Dengan ukuran batch 20 catatan dan durasi Fungsi Peristiwa target rata-rata 50 ms, setiap poller dapat menangani 2.000 catatan per detik dengan batas 5 MB/s . Ini dihitung sebagai: (20 catatan × 1000ms/50ms) × 5 pemanggilan Lambda bersamaan. Oleh karena itu, jika tingkat konsumsi puncak yang Anda inginkan adalah 20.000 catatan per detik, Anda memerlukan setidaknya 10 poller acara.

> **Note**
>
> Kami merekomendasikan untuk menyediakan poller acara tambahan sebagai penyangga agar tidak beroperasi secara konsisten pada kapasitas maksimum.

Mode yang disediakan secara otomatis menskalakan poller acara Anda berdasarkan pola lalu lintas dalam poller peristiwa minimum dan maksimum yang dikonfigurasi yang dapat memicu penyeimbangan kembali, dan karenanya, memperkenalkan latensi tambahan. Anda dapat menonaktifkan auto-scaling dengan mengonfigurasi nilai yang sama untuk poller peristiwa minimum dan maksimum.

Pertimbangan tambahan

Beberapa pertimbangan tambahan meliputi:

- Dingin dimulai dari pemanggilan fungsi target Lambda Anda berpotensi meningkatkan latensi ujung ke ujung. Untuk mengurangi risiko ini, pertimbangkan untuk mengaktifkan konkurensi yang disediakan atau SnapStartpada fungsi target pemetaan sumber peristiwa Anda. Selain itu, optimalkan alokasi memori fungsi Anda untuk memastikan eksekusi yang konsisten dan optimal.
- Ketika `MaximumBatchingWindowInSeconds` diatur ke 0, Lambda segera memproses catatan yang tersedia tanpa menunggu untuk mengisi ukuran batch lengkap. Misalnya, jika ukuran batch Anda disetel ke 1.000 catatan tetapi hanya 100 catatan yang tersedia, Lambda memproses 100 catatan tersebut dengan segera daripada menunggu 1.000 catatan penuh terakumulasi.

> **Important**
>
> Konfigurasi optimal untuk pemrosesan latensi rendah bervariasi secara signifikan berdasarkan beban kerja spesifik Anda. Kami sangat menyarankan untuk menguji konfigurasi yang berbeda dengan beban kerja Anda yang sebenarnya untuk menentukan pengaturan terbaik untuk kasus penggunaan Anda.

Mengonfigurasi kontrol penanganan kesalahan untuk sumber acara Kafka

Anda dapat mengonfigurasi cara Lambda menangani kesalahan dan percobaan ulang untuk pemetaan sumber acara Kafka Anda. Konfigurasi ini membantu Anda mengontrol cara Lambda memproses catatan yang gagal dan mengelola perilaku coba lagi.

Konfigurasi coba lagi yang tersedia

Konfigurasi coba lagi berikut tersedia untuk Amazon MSK dan sumber acara Kafka yang dikelola sendiri:

- Upaya coba ulang maksimum - Jumlah maksimum kali Lambda mencoba ulang saat Fungsi Acara Anda mengembalikan kesalahan. Ini tidak menghitung upaya pemanggilan awal. Defaultnya adalah -1 (tak terbatas). Saat Anda mengonfigurasi percobaan ulang tak terbatas dan tujuan saat gagal, Lambda secara otomatis menerapkan maksimal 10 percobaan ulang.
- Usia rekaman maksimum - Usia maksimum rekaman yang dikirim Lambda ke fungsi Anda. Defaultnya adalah -1 (tak terbatas).
- Pisahkan batch pada kesalahan - Ketika fungsi Anda mengembalikan kesalahan, bagi batch menjadi dua batch yang lebih kecil dan coba lagi masing-masing secara terpisah. Ini membantu mengisolasi catatan bermasalah.
- Respons batch sebagian - Izinkan fungsi Anda mengembalikan informasi tentang catatan mana dalam batch yang gagal diproses, sehingga Lambda hanya dapat mencoba kembali catatan yang gagal.

Mengkonfigurasi kontrol penanganan kesalahan (konsol)

Anda dapat mengonfigurasi perilaku coba lagi saat membuat atau memperbarui pemetaan sumber peristiwa Kafka di konsol Lambda.

Untuk mengonfigurasi perilaku coba lagi untuk sumber acara Kafka (konsol)

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih nama fungsi Anda.
3. Lakukan salah satu tindakan berikut:
   - Untuk menambahkan pemicu Kafka baru, di bawah Ikhtisar fungsi, pilih Tambah pemicu.
   - Untuk memodifikasi pemicu Kafka yang ada, pilih pemicu dan kemudian pilih Edit.

4. Di bawah konfigurasi poller peristiwa, pilih mode yang disediakan untuk mengonfigurasi kontrol penanganan kesalahan:
   a. Untuk mencoba lagi, masukkan jumlah maksimum percobaan ulang (0-10000, atau -1 untuk tak terbatas).
   b. Untuk usia rekaman maksimum, masukkan usia maksimum dalam hitungan detik (60-604800, atau -1 untuk tak terbatas).
   c. Untuk mengaktifkan pemisahan batch saat terjadi kesalahan, pilih Split batch on error.
   d. Untuk mengaktifkan respons batch sebagian, pilih ReportBatchItemFailures.
5. Pilih Tambah atau Simpan.

Mengkonfigurasi perilaku coba lagi (AWS CLI)

Gunakan AWS CLI perintah berikut untuk mengonfigurasi perilaku coba lagi untuk pemetaan sumber acara Kafka Anda.

Membuat pemetaan sumber peristiwa dengan konfigurasi coba lagi

Contoh berikut membuat pemetaan sumber peristiwa Kafka yang dikelola sendiri dengan kontrol penanganan kesalahan:

```
aws lambda create-event-source-mapping \
  --function-name my-kafka-function \
  --topics my-kafka-topic \
  --source-access-configuration Type=SASL_SCRAM_512_AUTH,URI=arn:aws:secretsmanager:us-
east-1:111122223333:secret:MyBrokerSecretName \
  --self-managed-event-source '{"Endpoints":{"KAFKA_BOOTSTRAP_SERVERS":
["abc.xyz.com:9092"]}}' \
  --starting-position LATEST \
  --provisioned-poller-config MinimumPollers=1,MaximumPollers=1 \
  --maximum-retry-attempts 3 \
  --maximum-record-age-in-seconds 3600 \
  --bisect-batch-on-function-error \
  --function-response-types "ReportBatchItemFailures"
```

Untuk sumber acara MSK Amazon:

```
aws lambda create-event-source-mapping \
  --event-source-arn arn:aws:kafka:us-east-1:111122223333:cluster/my-cluster/fc2f5bdf-
fd1b-45ad-85dd-15b4a5a6247e-2 \
```

```
  --topics AWSMSKKafkaTopic \
  --starting-position LATEST \
  --function-name my-kafka-function \
  --source-access-configurations '[{"Type": "SASL_SCRAM_512_AUTH","URI":
 "arn:aws:secretsmanager:us-east-1:111122223333:secret:my-secret"}]' \
  --provisioned-poller-config MinimumPollers=1,MaximumPollers=1 \
  --maximum-retry-attempts 3 \
  --maximum-record-age-in-seconds 3600 \
  --bisect-batch-on-function-error \
  --function-response-types "ReportBatchItemFailures"
```

Memperbarui konfigurasi coba lagi

Gunakan `update-event-source-mapping` perintah untuk memodifikasi konfigurasi coba lagi untuk pemetaan sumber peristiwa yang ada:

```
aws lambda update-event-source-mapping \
  --uuid 12345678-1234-1234-1234-123456789012 \
  --maximum-retry-attempts 5 \
  --maximum-record-age-in-seconds 7200 \
  --bisect-batch-on-function-error \
  --function-response-types "ReportBatchItemFailures"
```

PartialBatchResponse

Respons batch sebagian, juga dikenal sebagai ReportBatchItemFailures, adalah fitur kunci untuk penanganan kesalahan dalam integrasi Lambda dengan sumber Kafka. Tanpa fitur ini, ketika terjadi kesalahan di salah satu item dalam batch, itu menghasilkan pemrosesan ulang semua pesan dalam batch itu. Dengan respons batch sebagian diaktifkan dan diimplementasikan, handler mengembalikan pengenal hanya untuk pesan yang gagal, memungkinkan Lambda untuk mencoba lagi hanya item tertentu tersebut. Ini memberikan kontrol yang lebih besar atas bagaimana batch yang berisi pesan gagal diproses.

Untuk melaporkan kesalahan batch, gunakan skema JSON ini:

```
{
  "batchItemFailures": [
    {
      "itemIdentifier": {
        "partition": "topic-partition_number",
        "offset": 100
```

```
      }
    },
    ...
  ]
}
```

> **Important**
>
> Jika Anda mengembalikan JSON atau null yang valid kosong, pemetaan sumber peristiwa menganggap batch berhasil diproses. Setiap topic-partition_number atau offset yang tidak valid yang dikembalikan yang tidak ada dalam acara yang dipanggil diperlakukan sebagai kegagalan dan seluruh batch dicoba ulang.

Contoh kode berikut menunjukkan cara mengimplementasikan respons batch sebagian untuk Fungsi Peristiwa Lambda yang menerima peristiwa dari sumber Kafka. Fungsi melaporkan kegagalan item batch dalam respons, memberi sinyal ke Lambda untuk mencoba lagi pesan tersebut nanti.

Berikut adalah implementasi handler Lambda Python yang menunjukkan pendekatan ini:

```
import base64
from typing import Any, Dict, List

def lambda_handler(event: Dict[str, Any], context: Any) -> Dict[str, List[Dict[str,
 Dict[str, Any]]]]:
    failures: List[Dict[str, Dict[str, Any]]] = []
    records_dict = event.get("records", {})

    for topic_partition, records_list in records_dict.items():
        for record in records_list:
            topic = record.get("topic")
            partition = record.get("partition")
            offset = record.get("offset")
            value_b64 = record.get("value")

            try:
                data = base64.b64decode(value_b64).decode("utf-8")
                process_message(data)
            except Exception as exc:
                print(f"Failed to process record topic={topic} partition={partition}
 offset={offset}: {exc}")
                item_identifier: Dict[str, Any] = {
```

```
                    "partition": f"{topic}-{partition}",
                    "offset": int(offset) if offset is not None else None,
                }
                failures.append({"itemIdentifier": item_identifier})

    return {"batchItemFailures": failures}

def process_message(data: str) -> None:
    # Your business logic for a single message
    pass
```

Ini Node.js versinya:

```
const { Buffer } = require("buffer");

const handler = async (event) => {
  const failures = [];

  for (let topicPartition in event.records) {
    const records = event.records[topicPartition];

    for (const record of records) {
      const topic = record.topic;
      const partition = record.partition;
      const offset = record.offset;
      const valueBase64 = record.value;
      const data = Buffer.from(valueBase64, "base64").toString("utf8");

      try {
        await processMessage(data);
      } catch (error) {
        console.error("Failed to process record", { topic, partition, offset, error });
        const itemIdentifier = {
          "partition": `${topic}-${partition}`,
          "offset": Number(offset),
        };
        failures.push({ itemIdentifier });
      }
    }
  }

  return { batchItemFailures: failures };
};
```

```
async function processMessage(payload) {
  // Your business logic for a single message
}

module.exports = { handler };
```

Berikut adalah versi Java:

```
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;

import java.util.ArrayList;
import java.util.Base64;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class KafkaBatchHandler implements RequestHandler<Map<String, Object>,
 Map<String, Object>> {

    @SuppressWarnings("unchecked")
    @Override
    public Map<String, Object> handleRequest(Map<String, Object> event, Context
 context) {
        List<Map<String, Object>> failures = new ArrayList<>();
        Map<String, List<Map<String, Object>>> records =
                (Map<String, List<Map<String, Object>>>) event.getOrDefault("records",
 Map.of());

        for (Map.Entry<String, List<Map<String, Object>>> entry : records.entrySet()) {
            for (Map<String, Object> record : entry.getValue()) {
                String topic = (String) record.get("topic");
                Object partition = record.get("partition");
                Object offset = record.get("offset");
                String valueBase64 = (String) record.get("value");

                try {
                    String data = new String(Base64.getDecoder().decode(valueBase64),
 "UTF-8");
                    processMessage(data);
                } catch (Exception e) {
```

```
                    System.err.printf("Failed to process record topic=%s partition=%s
 offset=%s: %s%n",
                            topic, partition, offset, e.getMessage());
                    Map<String, Object> itemIdentifier = new HashMap<>();
                    itemIdentifier.put("partition", topic + "-" + partition);
                    itemIdentifier.put("offset", offset instanceof Number ? ((Number)
 offset).longValue() : null);
                    Map<String, Object> failure = new HashMap<>();
                    failure.put("itemIdentifier", itemIdentifier);
                    failures.add(failure);
                }
            }
        }

        Map<String, Object> response = new HashMap<>();
        response.put("batchItemFailures", failures);
        return response;
    }

    private void processMessage(String data) {
        // Your business logic for a single message
    }
}
```

Menangkap batch yang dibuang untuk Amazon MSK dan sumber acara Apache Kafka yang dikelola sendiri

Untuk menyimpan catatan pemanggilan pemetaan sumber peristiwa yang gagal, tambahkan tujuan ke pemetaan sumber peristiwa Fungsi Acara Anda. Setiap catatan yang dikirim ke tujuan adalah dokumen JSON yang berisi metadata tentang pemanggilan yang gagal. Untuk tujuan Amazon S3, Lambda juga mengirimkan seluruh catatan pemanggilan bersama dengan metadata. Anda dapat mengonfigurasi topik Amazon SNS, antrian Amazon SQS, bucket Amazon S3, atau Kafka sebagai tujuan.

Dengan tujuan Amazon S3, Anda dapat menggunakan fitur Pemberitahuan Acara Amazon S3 untuk menerima notifikasi saat objek diunggah ke bucket S3 tujuan Anda. Anda juga dapat mengonfigurasi Pemberitahuan Acara S3 untuk memanggil Fungsi Peristiwa Lambda lain untuk melakukan pemrosesan otomatis pada batch yang gagal.

Peran eksekusi Anda harus memiliki izin untuk tujuan:

- Untuk tujuan SQS: sqs: SendMessage
- Untuk tujuan SNS: SNS: Publish
- Untuk tujuan S3: s3: PutObject dan s3: ListBucket
- Untuk tujuan Kafka: kafka-cluster: WriteData

Anda dapat mengonfigurasi topik Kafka sebagai tujuan gagal untuk pemetaan sumber acara Kafka Anda. Ketika Lambda tidak dapat memproses catatan setelah percobaan ulang yang melelahkan atau ketika catatan melebihi usia maksimum, Lambda mengirimkan catatan yang gagal ke topik Kafka yang ditentukan untuk diproses nanti. Lihat the section called “Tujuan kegagalan Kafka”.

Anda harus menerapkan titik akhir VPC untuk layanan tujuan yang gagal di dalam VPC cluster Kafka Anda.

Selain itu, jika Anda mengonfigurasi kunci KMS di tujuan Anda, Lambda memerlukan izin berikut tergantung pada jenis tujuan:

- Jika Anda telah mengaktifkan enkripsi dengan kunci KMS Anda sendiri untuk tujuan S3, kms: GenerateDataKey diperlukan. Jika kunci KMS dan tujuan bucket S3 berada di akun yang berbeda dari Fungsi Peristiwa Lambda dan peran eksekusi, konfigurasikan kunci KMS untuk mempercayai peran eksekusi agar memungkinkan kms:. GenerateDataKey

- Jika Anda telah mengaktifkan enkripsi dengan kunci KMS Anda sendiri untuk tujuan SQS, KMS: Decrypt dan kms: diperlukan. GenerateDataKey Jika kunci KMS dan tujuan antrean SQS berada di akun yang berbeda dari Fungsi Peristiwa Lambda dan peran eksekusi, konfigurasikan kunci KMS untuk mempercayai peran eksekusi agar memungkinkan KMS:Decrypt, kms:, kms:, dan kms:. GenerateDataKey DescribeKey ReEncrypt
- Jika Anda telah mengaktifkan enkripsi dengan kunci KMS Anda sendiri untuk tujuan SNS, KMS: Decrypt dan kms: diperlukan. GenerateDataKey Jika kunci KMS dan tujuan topik SNS berada di akun yang berbeda dari Fungsi Peristiwa Lambda dan peran eksekusi, konfigurasikan kunci KMS untuk mempercayai peran eksekusi untuk mengizinkan KMS:Decrypt, kms:, kms:, dan kms:. GenerateDataKey DescribeKey ReEncrypt

Mengonfigurasi tujuan yang gagal untuk pemetaan sumber acara Kafka

Untuk mengonfigurasi tujuan saat gagal menggunakan konsol, ikuti langkah-langkah berikut:

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih fungsi.
3. Di bagian Gambaran umum fungsi, pilih Tambahkan tujuan.
4. Untuk Sumber, pilih Pemanggilan pemetaan sumber acara.
5. Untuk pemetaan sumber peristiwa, pilih sumber peristiwa yang dikonfigurasi untuk fungsi ini.
6. Untuk Kondisi, pilih On failure. Untuk pemanggilan pemetaan sumber peristiwa, ini adalah satu-satunya kondisi yang diterima.
7. Untuk tipe Tujuan, pilih tipe tujuan yang Lambda kirimkan catatan pemanggilan.
8. Untuk Tujuan, pilih sumber daya.
9. Pilih Simpan.

Anda juga dapat mengonfigurasi tujuan pada kegagalan menggunakan file. AWS CLI Misalnya, perintah create-event-source-mapping berikut menambahkan pemetaan sumber peristiwa dengan tujuan kegagalan SQS ke: `MyFunction`

```
aws lambda create-event-source-mapping \
--function-name "MyFunction" \
--event-source-arn arn:aws:kafka:us-east-1:123456789012:cluster/
vpc-2priv-2pub/751d2973-a626-431c-9d4e-d7975eb44dd7-2 \
```

```
--destination-config '{"OnFailure": {"Destination": "arn:aws:sqs:us-
east-1:123456789012:dest-queue"}}'
```

Perintah update-event-source-mapping berikut menambahkan tujuan kegagalan S3 ke sumber peristiwa yang terkait dengan input: `uuid`

```
aws lambda update-event-source-mapping \
--uuid f89f8514-cdd9-4602-9e1f-01a5b77d449b \
--destination-config '{"OnFailure": {"Destination": "arn:aws:s3:::dest-bucket"}}'
```

Untuk menghapus tujuan, berikan string kosong sebagai argumen ke `destination-config` parameter:

```
aws lambda update-event-source-mapping \
--uuid f89f8514-cdd9-4602-9e1f-01a5b77d449b \
--destination-config '{"OnFailure": {"Destination": ""}}'
```

Praktik terbaik keamanan untuk tujuan Amazon S3

Menghapus bucket S3 yang dikonfigurasi sebagai tujuan tanpa menghapus tujuan dari konfigurasi fungsi Anda dapat menimbulkan risiko keamanan. Jika pengguna lain mengetahui nama bucket tujuan Anda, mereka dapat membuat ulang bucket di bucket tersebut. Akun AWS Catatan pemanggilan yang gagal akan dikirim ke bucket mereka, yang berpotensi mengekspos data dari fungsi Anda.

> **Warning**
>
> Untuk memastikan bahwa catatan pemanggilan dari Fungsi Peristiwa Anda tidak dapat dikirim ke bucket S3 di bucket lain Akun AWS, tambahkan kondisi ke peran eksekusi fungsi Anda yang membatasi `s3:PutObject` izin ke bucket di akun Anda.

Contoh berikut menunjukkan kebijakan IAM yang membatasi `s3:PutObject` izin fungsi Anda ke bucket di akun Anda. Kebijakan ini juga memberi Lambda `s3:ListBucket` izin yang dibutuhkan untuk menggunakan bucket S3 sebagai tujuan.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
```

```
            "Sid": "S3BucketResourceAccountWrite",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::*/*",
                "arn:aws:s3:::*"
            ],
            "Condition": {
                "StringEquals": {
                    "s3:ResourceAccount": "111122223333"
                }
            }
        }
    ]
}
```

Untuk menambahkan kebijakan izin ke peran eksekusi fungsi Anda menggunakan Konsol Manajemen AWS atau AWS CLI, lihat instruksi dalam prosedur berikut:

Console

Untuk menambahkan kebijakan izin ke peran eksekusi fungsi (konsol)

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih Lambda Event Function yang peran eksekusinya ingin Anda ubah.
3. Di tab Konfigurasi, pilih Izin.
4. Di tab Peran eksekusi, pilih nama Peran fungsi Anda untuk membuka halaman konsol IAM peran.
5. Tambahkan kebijakan izin ke peran dengan melakukan hal berikut:
    a. Di panel Kebijakan izin, pilih Tambahkan izin, lalu pilih Buat kebijakan sebaris.
    b. Di Editor kebijakan, pilih JSON.
    c. Rekatkan kebijakan yang ingin Anda tambahkan ke editor (ganti JSON yang ada), lalu pilih Berikutnya.
    d. Di bawah Detail kebijakan, masukkan nama Kebijakan.
    e. Pilih Buat kebijakan.

AWS CLI

Untuk menambahkan kebijakan izin ke peran eksekusi Fungsi Peristiwa (CLI)

1. Buat dokumen kebijakan JSON dengan izin yang diperlukan dan simpan di direktori lokal.
2. Gunakan perintah IAM `put-role-policy` CLI untuk menambahkan izin ke peran eksekusi fungsi Anda. Jalankan perintah berikut dari direktori tempat Anda menyimpan dokumen kebijakan JSON dan ganti nama peran, nama kebijakan, dan dokumen kebijakan dengan nilai Anda sendiri.

```
aws iam put-role-policy \
--role-name my_lambda_role \
--policy-name LambdaS3DestinationPolicy \
--policy-document file://my_policy.json
```

Catatan pemanggilan contoh SNS dan SQS

Contoh berikut menunjukkan apa yang Lambda kirim ke topik SNS atau tujuan antrian SQS untuk pemanggilan sumber peristiwa Kafka yang gagal. Setiap kunci di bawah `recordsInfo` berisi topik dan partisi Kafka, dipisahkan oleh tanda hubung. Misalnya, untuk kuncinya`"Topic-0"`, `Topic` adalah topik Kafka, dan `0` merupakan partisi. Untuk setiap topik dan partisi, Anda dapat menggunakan data offset dan stempel waktu untuk menemukan catatan pemanggilan asli.

```
{
    "requestContext": {
        "requestId": "316aa6d0-8154-xmpl-9af7-85d5f4a6bc81",
        "functionArn": "arn:aws:lambda:us-east-1:123456789012:function:myfunction",
        "condition": "RetryAttemptsExhausted" | "MaximumPayloadSizeExceeded",
        "approximateInvokeCount": 1
    },
    "responseContext": { // null if record is MaximumPayloadSizeExceeded
        "statusCode": 200,
        "executedVersion": "$LATEST",
        "functionError": "Unhandled"
    },
    "version": "1.0",
    "timestamp": "2019-11-14T00:38:06.021Z",
    "KafkaBatchInfo": {
        "batchSize": 500,
```

```
        "eventSourceArn": "arn:aws:kafka:us-east-1:123456789012:cluster/
vpc-2priv-2pub/751d2973-a626-431c-9d4e-d7975eb44dd7-2",
        "bootstrapServers": "...",
        "payloadSize": 2039086, // In bytes
        "recordsInfo": {
            "Topic-0": {
                "firstRecordOffset":
 "49601189658422359378836298521827638475320189012309704722",
                "lastRecordOffset":
 "49601189658422359378836298522902373528957594348623495186",
                "firstRecordTimestamp": "2019-11-14T00:38:04.835Z",
                "lastRecordTimestamp": "2019-11-14T00:38:05.580Z",
            },
            "Topic-1": {
                "firstRecordOffset":
 "49601189658422359378836298521827638475320189012309704722",
                "lastRecordOffset":
 "49601189658422359378836298522902373528957594348623495186",
                "firstRecordTimestamp": "2019-11-14T00:38:04.835Z",
                "lastRecordTimestamp": "2019-11-14T00:38:05.580Z",
            }
        }
    }
}
```

Catatan pemanggilan contoh tujuan S3

Untuk tujuan S3, Lambda mengirimkan seluruh catatan pemanggilan bersama dengan metadata ke tujuan. Contoh berikut menunjukkan bahwa Lambda mengirim ke tujuan bucket S3 untuk pemanggilan sumber peristiwa Kafka yang gagal. Selain semua bidang dari contoh sebelumnya untuk tujuan SQS dan SNS, `payload` bidang berisi catatan pemanggilan asli sebagai string JSON yang lolos.

```
{
    "requestContext": {
        "requestId": "316aa6d0-8154-xmpl-9af7-85d5f4a6bc81",
        "functionArn": "arn:aws:lambda:us-east-1:123456789012:function:myfunction",
        "condition": "RetryAttemptsExhausted" | "MaximumPayloadSizeExceeded",
        "approximateInvokeCount": 1
    },
    "responseContext": { // null if record is MaximumPayloadSizeExceeded
        "statusCode": 200,
        "executedVersion": "$LATEST",
```

```
        "functionError": "Unhandled"
    },
    "version": "1.0",
    "timestamp": "2019-11-14T00:38:06.021Z",
    "KafkaBatchInfo": {
        "batchSize": 500,
        "eventSourceArn": "arn:aws:kafka:us-east-1:123456789012:cluster/
vpc-2priv-2pub/751d2973-a626-431c-9d4e-d7975eb44dd7-2",
        "bootstrapServers": "...",
        "payloadSize": 2039086, // In bytes
        "recordsInfo": {
            "Topic-0": {
                "firstRecordOffset":
 "49601189658422359378836298521827638475320189012309704722",
                "lastRecordOffset":
 "49601189658422359378836298522902373528957594348623495186",
                "firstRecordTimestamp": "2019-11-14T00:38:04.835Z",
                "lastRecordTimestamp": "2019-11-14T00:38:05.580Z",
            },
            "Topic-1": {
                "firstRecordOffset":
 "49601189658422359378836298521827638475320189012309704722",
                "lastRecordOffset":
 "49601189658422359378836298522902373528957594348623495186",
                "firstRecordTimestamp": "2019-11-14T00:38:04.835Z",
                "lastRecordTimestamp": "2019-11-14T00:38:05.580Z",
            }
        }
    },
    "payload": "<Whole Event>" // Only available in S3
}
```

> **Tip**
>
> Sebaiknya aktifkan versi S3 di bucket tujuan Anda.

Menggunakan topik Kafka sebagai tujuan kegagalan

Anda dapat mengonfigurasi topik Kafka sebagai tujuan gagal untuk pemetaan sumber acara Kafka Anda. Ketika Lambda tidak dapat memproses catatan setelah percobaan ulang yang melelahkan atau ketika catatan melebihi usia maksimum, Lambda mengirimkan catatan yang gagal ke topik Kafka yang ditentukan untuk diproses nanti. Saat Anda mengonfigurasi percobaan ulang tak terbatas dan tujuan saat gagal, Lambda secara otomatis menerapkan maksimal 10 percobaan ulang.

Cara kerja tujuan on-failure Kafka

Saat Anda mengonfigurasi topik Kafka sebagai tujuan yang gagal, Lambda bertindak sebagai produser Kafka dan menulis catatan gagal ke topik tujuan. Ini menciptakan pola topik surat mati (DLT) dalam infrastruktur Kafka Anda.

- Persyaratan cluster yang sama - Topik tujuan harus ada di cluster Kafka yang sama dengan topik sumber Anda.
- Konten rekaman aktual — Tujuan Kafka menerima catatan gagal aktual bersama dengan metadata kegagalan.
- Pencegahan rekursi — Lambda mencegah loop tak terbatas dengan memblokir konfigurasi di mana topik sumber dan tujuan sama.

Otentikasi untuk tujuan Kafka

Tujuan kegagalan Kafka tidak memiliki pengaturan otentikasi sendiri. Topik tujuan berada dalam klaster yang sama dengan topik sumber Anda, jadi Lambda menulis catatan gagal dengan otentikasi yang telah Anda konfigurasikan dalam `SourceAccessConfigurations` parameter pemetaan sumber peristiwa.

Ini berlaku untuk setiap metode otentikasi yang didukung Lambda untuk Kafka, termasuk OAuth 2.0 `OAUTHBEARER_AUTH` (), IAM (), dan IAM with `IAM_AUTH` (). SASL/OAUTHBEARER `IAM_OAUTHBEARER_AUTH` Jika Anda mengautentikasi dengan OAuth 2.0, Lambda menggunakan penyedia identitas yang sama dan rahasia yang sama untuk mendapatkan token untuk memproduksi ke topik tujuan. Konfigurasikan otentikasi sekali, dalam konfigurasi akses sumber. Untuk informasi selengkapnya, lihat the section called “Autentikasi klaster” Kafka yang dikelola sendiri dan the section called “Autentikasi klaster” Amazon MSK.

Pastikan identitas yang Anda autentikasi dapat menulis ke topik tujuan, tidak hanya dibaca dari topik sumber Anda. Untuk otentikasi IAM, itu berarti `kafka-cluster:WriteData` pada topik tujuan. Untuk metode lain, itu berarti izin tulis yang setara dalam aturan otorisasi klaster Anda.

Mengonfigurasi tujuan pada kegagalan Kafka

Anda dapat mengonfigurasi topik Kafka sebagai tujuan gagal saat membuat atau memperbarui pemetaan sumber acara Kafka.

Mengkonfigurasi tujuan Kafka (konsol)

Untuk mengonfigurasi topik Kafka sebagai tujuan gagal (konsol)

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih nama fungsi Anda.
3. Lakukan salah satu tindakan berikut:
   - Untuk menambahkan pemicu Kafka baru, di bawah Ikhtisar fungsi, pilih Tambah pemicu.
   - Untuk memodifikasi pemicu Kafka yang ada, pilih pemicu dan kemudian pilih Edit.
4. Di bawah Pengaturan tambahan, untuk On-failure tujuan, pilih topik Kafka.
5. Untuk nama Topik, masukkan nama topik Kafka tempat Anda ingin mengirim catatan yang gagal.
6. Pilih Tambah atau Simpan.

Mengkonfigurasi tujuan Kafka (AWS CLI)

Gunakan `kafka://` awalan untuk menentukan topik Kafka sebagai tujuan kegagalan.

Membuat pemetaan sumber acara dengan tujuan Kafka

Contoh berikut membuat pemetaan sumber peristiwa MSK Amazon dengan topik Kafka sebagai tujuan kegagalan:

```
aws lambda create-event-source-mapping \
  --function-name my-kafka-function \
  --topics AWSKafkaTopic \
  --event-source-arn arn:aws:kafka:us-east-1:123456789012:cluster/my-cluster/abc123 \
  --starting-position LATEST \
  --provisioned-poller-config MinimumPollers=1,MaximumPollers=3 \
  --destination-config '{"OnFailure":{"Destination":"kafka://failed-records-topic"}}'
```

Untuk Kafka yang dikelola sendiri, gunakan sintaks yang sama:

```
aws lambda create-event-source-mapping \
```

```
  --function-name my-kafka-function \
  --topics AWSKafkaTopic \
  --self-managed-event-source '{"Endpoints":{"KAFKA_BOOTSTRAP_SERVERS":
["abc.xyz.com:9092"]}}' \
  --starting-position LATEST \
  --provisioned-poller-config MinimumPollers=1,MaximumPollers=3 \
  --destination-config '{"OnFailure":{"Destination":"kafka://failed-records-topic"}}'
```

Memperbarui tujuan Kafka

Gunakan `update-event-source-mapping` perintah untuk menambah atau memodifikasi tujuan Kafka:

```
aws lambda update-event-source-mapping \
  --uuid 12345678-1234-1234-1234-123456789012 \
  --destination-config '{"OnFailure":{"Destination":"kafka://failed-records-topic"}}'
```

Format rekaman untuk tujuan Kafka

Ketika Lambda mengirim catatan gagal ke topik Kafka, setiap pesan berisi metadata tentang kegagalan dan konten rekaman yang sebenarnya.

Metadata kegagalan

Metadata mencakup informasi tentang mengapa catatan gagal dan detail tentang batch asli:

```
{
  "requestContext": {
    "requestId": "e4b46cbf-b738-xmpl-8880-a18cdf61200e",
    "functionArn": "arn:aws:lambda:us-east-1:123456789012:function:my-function:
$LATEST",
    "condition": "RetriesExhausted",
    "approximateInvokeCount": 3
  },
  "responseContext": {
    "statusCode": 200,
    "executedVersion": "$LATEST",
    "functionError": "Unhandled"
  },
  "version": "1.0",
  "timestamp": "2019-11-14T18:16:05.568Z",
  "KafkaBatchInfo": {
    "batchSize": 1,
```

```
    "eventSourceArn": "arn:aws:kafka:us-east-1:123456789012:cluster/my-cluster/abc123",
    "bootstrapServers": "b-1.mycluster.abc123.kafka.us-east-1.amazonaws.com:9098",
    "payloadSize": 1162,
    "recordInfo": {
      "offset": "4960118965842235937883629852182763847532018901230970472 2",
      "timestamp": "2019-11-14T18:16:04.835Z"
    }
  },
  "payload": {
    "bootstrapServers": "b-1.mycluster.abc123.kafka.us-east-1.amazonaws.com:9098",
    "eventSource": "aws:kafka",
    "eventSourceArn": "arn:aws:kafka:us-east-1:123456789012:cluster/my-cluster/abc123",
    "records": {
      "my-topic-0": [
        {
          "headers": [],
          "key": "dGVzdC1rZXk=",
          "offset": 100,
          "partition": 0,
          "timestamp": 1749116692330,
          "timestampType": "CREATE_TIME",
          "topic": "my-topic",
          "value": "dGVzdC12YWx1ZQ=="
        }
      ]
    }
  }
}
```

Perilaku kunci partisi

Lambda menggunakan kunci partisi yang sama dari catatan asli saat memproduksi ke topik tujuan. Jika rekaman asli tidak memiliki kunci, Lambda menggunakan partisi round-robin default Kafka di semua partisi yang tersedia di topik tujuan.

Persyaratan dan pembatasan

- Diperlukan mode yang disediakan - Tujuan saat kegagalan Kafka hanya tersedia untuk pemetaan sumber peristiwa dengan mode yang disediakan diaktifkan.
- Hanya cluster yang sama - Topik tujuan harus ada di cluster Kafka yang sama dengan topik sumber Anda.
- Izin topik - Pemetaan sumber acara Anda harus memiliki izin menulis ke topik tujuan. Contoh:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ClusterPermissions",
            "Effect": "Allow",
            "Action": [
                "kafka-cluster:Connect",
                "kafka-cluster:DescribeCluster",
                "kafka-cluster:DescribeTopic",
                "kafka-cluster:WriteData",
                "kafka-cluster:ReadData"
            ],
            "Resource": [
                "arn:aws:kafka:*:*:cluster/*"
            ]
        },
        {
            "Sid": "TopicPermissions",
            "Effect": "Allow",
            "Action": [
                "kafka-cluster:DescribeTopic",
                "kafka-cluster:WriteData",
                "kafka-cluster:ReadData"
            ],
            "Resource": [
                "arn:aws:kafka:*:*:topic/*/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "kafka:DescribeCluster",
                "kafka:GetBootstrapBrokers",
                "kafka:Produce"
            ],
            "Resource": "arn:aws:kafka:*:*:cluster/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterfaces",
```

```
                "ec2:DeleteNetworkInterface",
                "ec2:DescribeSubnets",
                "ec2:DescribeSecurityGroups"
            ],
            "Resource": "*"
        }
    ]
}
```

- Tidak ada rekursi - Nama topik tujuan tidak boleh sama dengan nama topik sumber Anda.

Pencatatan pemetaan sumber acara Kafka

Anda dapat mengonfigurasi pencatatan tingkat sistem untuk pemetaan sumber peristiwa Kafka Anda untuk mengaktifkan dan memfilter log sistem yang dikirim oleh poller acara Lambda. CloudWatch

Fitur ini hanya tersedia untuk pemetaan sumber acara Kafka, dan dengan mode Provisioned.

Untuk pemetaan sumber peristiwa dengan konfigurasi logging, Anda juga dapat memeriksa log sistem dari kueri log bawaan di tab Monitor dari halaman Konsol Lambda > Sumber daya tambahan> pemetaan sumber peristiwa sekarang.

Cara kerja logging

Saat Anda menyetel konfigurasi logging dengan level log dalam pemetaan sumber peristiwa, poller peristiwa Lambda mengirimkan log yang sesuai (log sistem pemetaan sumber peristiwa).

Pemetaan sumber peristiwa menggunakan kembali tujuan log yang sama dengan Fungsi Peristiwa Lambda Anda. Pastikan peran eksekusi Fungsi Peristiwa Lambda Anda memiliki izin logging yang diperlukan.

Pemetaan sumber peristiwa akan memiliki aliran lognya sendiri, dengan
tanggal dan sumber peristiwa pemetaan UUID sebagai nama aliran log, seperti.
`2020/01/01/12345678-1234-1234-1234-12345678901`

Untuk log sistem pemetaan sumber peristiwa, Anda dapat memilih antara tingkat log berikut.

Tingkat log	Penggunaan
DEBUG (paling detail)	Informasi terperinci untuk kemajuan pemrosesan sumber acara
INFO	Pesan tentang operasi normal pemetaan sumber acara
PERINGATAN (paling sedikit detail)	Pesan tentang peringatan potensial dan kesalahan yang mungkin menyebabkan perilaku tak terduga

Saat Anda memilih level log, poller acara Lambda mengirimkan log pada tingkat itu dan lebih rendah. Misalnya, jika Anda menyetel tingkat log sistem pemetaan sumber peristiwa ke INFO, poller peristiwa tidak mengirim output log pada tingkat DEBUG.

Mengonfigurasi pencatatan log

Anda dapat mengatur konfigurasi logging saat membuat atau memperbarui pemetaan sumber peristiwa Kafka.

Mengkonfigurasi logging (konsol)

Untuk mengkonfigurasi logging (konsol)

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih nama fungsi Anda.
3. Lakukan salah satu tindakan berikut:
   - Untuk menambahkan pemicu Kafka baru, di bawah Ikhtisar fungsi, pilih Tambah pemicu.
   - Untuk memodifikasi pemicu Kafka yang ada, pilih pemicu dan kemudian pilih Edit.
4. Di bawah konfigurasi poller Event, untuk mode Provisioned, aktifkan kotak centang Configure. Dan pengaturan level Log akan muncul.
5. Pilih daftar dropdown tingkat Log dan pilih level untuk pemetaan sumber acara.
6. Pilih Tambah atau Simpan di bagian bawah untuk membuat atau memperbarui pemetaan sumber acara.

Mengkonfigurasi logging (AWS CLI)

Membuat pemetaan sumber peristiwa dengan logging

Contoh berikut membuat pemetaan sumber peristiwa MSK Amazon dengan konfigurasi logging:

```
aws lambda create-event-source-mapping \
  --function-name my-kafka-function \
  --topics AWSKafkaTopic \
  --event-source-arn arn:aws:kafka:us-east-1:123456789012:cluster/my-cluster/abc123 \
  --starting-position LATEST \
  --provisioned-poller-config MinimumPollers=1,MaximumPollers=3 \
  --logging-config '{"SystemLogLevel":"DEBUG"}'
```

Untuk Kafka yang dikelola sendiri, gunakan sintaks yang sama:

```
aws lambda create-event-source-mapping \
  --function-name my-kafka-function \
  --topics AWSKafkaTopic \
  --self-managed-event-source '{"Endpoints":{"KAFKA_BOOTSTRAP_SERVERS":
["abc.xyz.com:9092"]}}' \
  --starting-position LATEST \
  --provisioned-poller-config MinimumPollers=1,MaximumPollers=3 \
  --logging-config '{"SystemLogLevel":"DEBUG"}'
```

Memperbarui konfigurasi logging

Gunakan `update-event-source-mapping` perintah untuk menambah atau memodifikasi konfigurasi logging:

```
aws lambda update-event-source-mapping \
  --uuid 12345678-1234-1234-1234-123456789012 \
  --logging-config '{"SystemLogLevel":"WARN"}'
```

Format rekaman untuk log sistem pemetaan sumber peristiwa Kafka

Ketika poller peristiwa Lambda mengirim log, setiap entri log berisi metadata pemetaan sumber peristiwa umum dan juga konten khusus acara.

PERINGATKAN catatan log

Catatan PERINGATAN berisi kesalahan atau peringatan dari poller acara, dan itu dipancarkan saat peristiwa terjadi. Contoh:

```
{
    "eventType": "ESM_PROCESSING_EVENT",
    "timestamp": 1546347650000,
    "resourceArn": "arn:aws:lambda:us-east-1:123456789012:event-source-
mapping:12345678-1234-1234-1234-123456789012",
    "eventSourceArn": "arn:aws:kafka:us-east-1:123456789012:cluster/tests-
cluster/87654321-4321-4321-4321-876543221-s1",
    "eventProcessorId": "12345678-1234-1234-1234-123456789012/0",
    "logLevel": "WARN",
    "error": {
        "errorMessage": "Timeout expired while fetching topic metadata",
        "errorCode": "org.apache.kafka.common.errors.TimeoutException"
```

```
    }
}
```

Catatan log INFO

Catatan INFO berisi konfigurasi klien konsumen Kafka di setiap poller acara, dan itu dipancarkan pada saat konsumen sedang dibangun atau diubah. Contoh:

```
{
    "eventType": "POLLER_STATUS_EVENT",
    "timestamp": 1546347660000,
    "resourceArn": "arn:aws:lambda:us-east-1:123456789012:event-source-
mapping:12345678-1234-1234-1234-123456789012",
    "eventSourceArn": "arn:aws:kafka:us-east-1:123456789012:cluster/tests-
cluster/87654321-4321-4321-4321-876543221-s1",
    "eventProcessorId": "12345678-1234-1234-1234-123456789012/0",
    "logLevel": "INFO",
    "kafkaEventSourceConnection": {
        "brokerEndpoints": "boot-abcd1234.c2.kafka-serverless.us-
east-1.amazonaws.com:9098",
        "consumerId": "12345678-1234-1234-1234-123456789012-0",
        "topics": [
            "test"
        ],
        "consumerGroupId": "12345678-1234-1234-1234-123456789012",
        "securityProtocol": "SASL_SSL",
        "saslMechanism": "AWS_MSK_IAM",
        "totalPartitionCount": 2,
        "assignedPartitionCount": 2,
        "partitionsAssignmentGeneration": 5,
        "assignedPartitions": [
            "test-0",
            "test-1"
        ],
        "networkConfig": {
            "ipAddresses": [
                "10.100.141.1"
            ],
            "subnetCidrBlock": "10.100.128.0/20",
            "securityGroups": [
                "sg-abcdefabcdefabcdef"
            ]
        }
```

```
    }
}
```

Catatan log DEBUG

Log DEBUG berisi info terkait offset Kafka dalam pemrosesan pemetaan sumber peristiwa, dan info offset dipancarkan per menit. Contoh:

```
{
    "eventType": "KAFKA_STATUS_EVENT",
    "timestamp": 1546347670000,
    "resourceArn": "arn:aws:lambda:us-east-1:123456789012:event-source-
mapping:12345678-1234-1234-1234-123456789012",
    "eventSourceArn": "arn:aws:kafka:us-east-1:123456789012:cluster/tests-
cluster/87654321-4321-4321-4321-876543221-s1",
    "eventProcessorId": "12345678-1234-1234-1234-123456789012/0",
    "logLevel": "DEBUG",
    "kafkaPartitionOffsets": {
        "partition": "test-1",
        "endOffset": 5004,
        "consumedOffset": 5003,
        "processedOffset": 5003,
        "committedOffset": 5004
    }
}
```

Memecahkan masalah kesalahan pemetaan sumber acara Kafka

Topik berikut memberikan saran pemecahan masalah untuk kesalahan dan masalah yang mungkin Anda temui saat menggunakan Amazon MSK atau Apache Kafka yang dikelola sendiri dengan Lambda.

Untuk bantuan lebih lanjut tentang pemecahan masalah, kunjungi Pusat AWS Pengetahuan.

Kesalahan otentikasi dan otorisasi

Jika salah satu izin yang diperlukan untuk mengkonsumsi data dari cluster Kafka tidak ada, Lambda menampilkan salah satu pesan kesalahan berikut dalam pemetaan sumber peristiwa di bawah. LastProcessingResult

Pesan kesalahan

- Cluster gagal mengotorisasi Lambda
- Otentikasi SASL gagal
- Server gagal mengautentikasi Lambda
- Lambda gagal mengautentikasi server
- Sertifikat atau kunci pribadi yang diberikan tidak valid

Cluster gagal mengotorisasi Lambda

Untuk SASL/SCRAM atau mTL, kesalahan ini menunjukkan bahwa pengguna yang disediakan tidak memiliki semua izin daftar kontrol akses (ACL) Kafka yang diperlukan berikut:

- DescribeConfigs Cluster
- Jelaskan Grup
- Baca Grup
- Jelaskan Topik
- Baca Topik

Saat Anda membuat ACL Kafka dengan `kafka-cluster` izin yang diperlukan, tentukan topik dan kelompokkan sebagai sumber daya. Nama topik harus cocok dengan topik dalam pemetaan sumber acara. Nama grup harus cocok dengan UUID pemetaan sumber peristiwa.

Setelah Anda menambahkan izin yang diperlukan ke peran eksekusi, mungkin perlu beberapa menit agar perubahan diterapkan.

Berikut ini adalah contoh log tingkat sistem ESM setelah mengaktifkan Logging Config untuk masalah ini:

```
{
    "eventType": "ESM_PROCESSING_EVENT",
    "timestamp": 1734567890123,
    "resourceArn": "arn:aws:lambda:us-east-1:123456789012:event-source-
mapping:a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
    "eventSourceArn": "arn:aws:kafka:us-east-1:123456789012:cluster/my-kafka-
cluster/12345678-abcd-1234-efgh-EXAMPLE11111-1",
    "eventProcessorId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111/0",
    "logLevel": "WARN",
    "error": {
        "errorMessage": "Not authorized to access topics: [my-topic]",
        "errorCode": "org.apache.kafka.common.errors.TopicAuthorizationException"
    }
}
```

Otentikasi SASL gagal

Untuk SASL/SCRAM atau SASL/PLAIN, kesalahan ini menunjukkan bahwa kredensyal masuk yang disediakan tidak valid.

Untuk kontrol akses IAM, peran eksekusi tidak memiliki `kafka-cluster:Connect` izin untuk cluster. Tambahkan izin ini ke peran dan tentukan Amazon Resource Name (ARN) cluster sebagai sumber daya.

Anda mungkin melihat kesalahan ini terjadi sebentar-sebentar. Cluster menolak koneksi setelah jumlah koneksi TCP melebihi kuota layanan. Lambda mundur dan mencoba lagi sampai koneksi berhasil. Setelah Lambda terhubung ke cluster dan polling untuk catatan, hasil pemrosesan terakhir berubah menjadi. `OK`

Berikut ini adalah contoh log tingkat sistem ESM setelah mengaktifkan Logging Config untuk masalah ini saat menggunakan otentikasi IAM:

```
{
    "eventType": "ESM_PROCESSING_EVENT",
    "timestamp": 1734567890456,
```

```
    "resourceArn": "arn:aws:lambda:us-east-1:123456789012:event-source-
mapping:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
    "eventSourceArn": "arn:aws:kafka:us-east-1:123456789012:cluster/my-kafka-
cluster/12345678-abcd-1234-efgh-EXAMPLE22222-1",
    "eventProcessorId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222/0",
    "logLevel": "WARN",
    "error": {
        "errorMessage": "[a1b2c3d4-5678-90ab-cdef-EXAMPLE22222]: Access denied",
        "errorCode": "org.apache.kafka.common.errors.SaslAuthenticationException"
    }
}
```

Server gagal mengautentikasi Lambda

Kesalahan ini menunjukkan bahwa broker Kafka gagal mengautentikasi Lambda. Ini dapat terjadi karena salah satu alasan berikut:

- Anda tidak memberikan sertifikat klien untuk otentikasi mTLS.
- Anda memberikan sertifikat klien, tetapi broker Kafka tidak dikonfigurasi untuk menggunakan otentikasi mTLS.
- Sertifikat klien tidak dipercaya oleh broker Kafka.

Lambda gagal mengautentikasi server

Kesalahan ini menunjukkan bahwa Lambda gagal mengautentikasi broker Kafka. Ini dapat terjadi karena salah satu alasan berikut:

- Untuk Apache Kafka yang dikelola sendiri: Broker Kafka menggunakan sertifikat yang ditandatangani sendiri atau CA pribadi, tetapi tidak memberikan sertifikat CA root server.
- Untuk Apache Kafka yang dikelola sendiri: Sertifikat CA root server tidak cocok dengan root CA yang menandatangani sertifikat broker.
- Validasi nama host gagal karena sertifikat broker tidak berisi nama DNS broker atau alamat IP sebagai nama alternatif subjek.

Sertifikat atau kunci pribadi yang diberikan tidak valid

Kesalahan ini menunjukkan bahwa konsumen Kafka tidak dapat menggunakan sertifikat atau kunci pribadi yang disediakan. Pastikan sertifikat dan kunci menggunakan format PEM, dan enkripsi kunci pribadi menggunakan algoritma PBES1.

Berikut ini adalah contoh log tingkat sistem ESM setelah mengaktifkan Logging Config untuk masalah ini:

```
{
    "eventType": "ESM_PROCESSING_EVENT",
    "timestamp": 1734567891234,
    "resourceArn": "arn:aws:lambda:us-east-1:123456789012:event-source-
mapping:a1b2c3d4-5678-90ab-cdef-EXAMPLE44444",
    "eventSourceArn": "arn:aws:kafka:us-east-1:123456789012:cluster/my-kafka-
cluster/12345678-abcd-1234-efgh-EXAMPLE44444-1",
    "eventProcessorId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE44444/0",
    "logLevel": "WARN",
    "error": {
        "errorMessage": "Invalid PEM keystore configs",
        "errorCode": "org.apache.kafka.common.errors.InvalidConfigurationException"
    }
}
```

Kesalahan jaringan dan konektivitas

Masalah konfigurasi jaringan dapat mencegah Lambda terhubung ke cluster Kafka Anda. Topik berikut menjelaskan kesalahan umum terkait jaringan.

Pesan kesalahan

- Batas waktu koneksi karena konfigurasi grup keamanan
- Titik akhir broker Kafka tidak dapat diselesaikan

Batas waktu koneksi karena konfigurasi grup keamanan

Jika grup keamanan yang terkait dengan cluster Kafka Anda tidak mengizinkan lalu lintas masuk dari dirinya sendiri, Lambda tidak dapat terhubung ke klaster. Pastikan aturan masuk grup keamanan memungkinkan lalu lintas dari grup keamanan itu sendiri di port broker Kafka.

Berikut ini adalah contoh log tingkat sistem ESM setelah mengaktifkan Logging Config untuk masalah ini:

```
{
    "eventType": "ESM_PROCESSING_EVENT",
    "timestamp": 1734567892345,
    "resourceArn": "arn:aws:lambda:us-east-1:123456789012:event-source-
mapping:a1b2c3d4-5678-90ab-cdef-EXAMPLE55555",
```

```
    "eventSourceArn": "arn:aws:kafka:us-east-1:123456789012:cluster/my-kafka-
cluster/12345678-abcd-1234-efgh-EXAMPLE55555-1",
    "eventProcessorId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE55555/0",
    "logLevel": "WARN",
    "error": {
        "errorMessage": "Timeout expired while fetching topic metadata",
        "errorCode": "org.apache.kafka.common.errors.TimeoutException"
    }
}
```

Anda juga dapat memeriksa log INFO konsumen Kafka untuk memverifikasi koneksi dan konfigurasi jaringan. `brokerEndpoints`Bidang menunjukkan alamat broker Kafka, `securityProtocol` dan `saslMechanism` (jika ada) menunjukkan metode otentikasi, dan `networkConfig` bidang menunjukkan alamat IP, blok CIDR subnet, dan grup keamanan yang digunakan oleh pemetaan sumber peristiwa. Verifikasi bahwa grup keamanan yang terdaftar mengizinkan lalu lintas masuk yang diperlukan:

```
{
    "eventType": "POLLER_STATUS_EVENT",
    "timestamp": 1734567892456,
    "resourceArn": "arn:aws:lambda:us-east-1:123456789012:event-source-
mapping:a1b2c3d4-5678-90ab-cdef-11111EXAMPLE",
    "eventSourceArn": "arn:aws:kafka:us-east-1:123456789012:cluster/my-kafka-cluster/
a1b2c3d4-5678-90ab-cdef-11111EXAMPLE-1",
    "eventProcessorId": "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE/0",
    "logLevel": "INFO",
    "kafkaEventSourceConnection": {
        "brokerEndpoints": "boot-abcd1234.c2.kafka-serverless.us-
east-1.amazonaws.com:9098",
        "consumerId": "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE-0",
        "topics": [
            "my-topic"
        ],
        "consumerGroupId": "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE",
        "securityProtocol": "SASL_SSL",
        "saslMechanism": "AWS_MSK_IAM",
        "totalPartitionCount": 2,
        "assignedPartitionCount": 2,
        "partitionsAssignmentGeneration": 1,
        "assignedPartitions": [
            "my-topic-0",
            "my-topic-1"
```

```
        ],
        "networkConfig": {
            "ipAddresses": [
                "10.0.0.37"
            ],
            "subnetCidrBlock": "10.0.0.32/28",
            "securityGroups": [
                "sg-0123456789abcdef0"
            ]
        }
    }
}
```

Titik akhir broker Kafka tidak dapat diselesaikan

Kesalahan ini menunjukkan bahwa cluster Kafka tidak ada atau telah dihapus. Verifikasi bahwa klaster yang ditentukan dalam pemetaan sumber peristiwa ada dan dalam keadaan aktif.

Berikut ini adalah contoh log tingkat sistem ESM setelah mengaktifkan Logging Config untuk masalah ini:

```
{
    "eventType": "ESM_PROCESSING_EVENT",
    "timestamp": 1734567893456,
    "resourceArn": "arn:aws:lambda:us-east-1:123456789012:event-source-
mapping:a1b2c3d4-5678-90ab-cdef-EXAMPLE66666",
    "eventSourceArn": "arn:aws:kafka:us-east-1:123456789012:cluster/my-kafka-
cluster/12345678-abcd-1234-efgh-EXAMPLE66666-1",
    "eventProcessorId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE66666/0",
    "logLevel": "WARN",
    "error": {
        "errorMessage": "No resolvable bootstrap urls given in bootstrap.servers",
        "errorCode": "org.apache.kafka.common.config.ConfigException"
    }
}
```

Kesalahan pemetaan sumber peristiwa

Saat Anda menambahkan cluster Apache Kafka sebagai sumber peristiwa untuk Fungsi Acara Lambda Anda, jika fungsi Anda mengalami kesalahan, konsumen Kafka Anda berhenti memproses catatan. Konsumen partisi topik adalah mereka yang berlangganan, membaca, dan memproses

catatan Anda. Konsumen Kafka Anda lainnya dapat melanjutkan pemrosesan catatan, asalkan tidak mengalami kesalahan yang sama.

Untuk menentukan penyebab penghentian konsumen, periksa bidang `StateTransitionReason` dalam respons dari `EventSourceMapping`. Daftar berikut menjelaskan kesalahan sumber peristiwa yang dapat Anda terima:

`ESM_CONFIG_NOT_VALID`

Konfigurasi pemetaan sumber peristiwa tidak valid.

`EVENT_SOURCE_AUTHN_ERROR`

Lambda tidak dapat mengautentikasi sumber acara.

`EVENT_SOURCE_AUTHZ_ERROR`

Lambda tidak memiliki izin yang diperlukan untuk mengakses sumber acara.

`FUNCTION_CONFIG_NOT_VALID`

Konfigurasi fungsi tidak valid.

> **Note**
>
> Jika catatan peristiwa Lambda Anda melebihi batas ukuran yang diizinkan sebesar 6 MB, catatan dapat tidak diproses.

Memanggil Fungsi Peristiwa Lambda menggunakan titik akhir Amazon API Gateway

Anda dapat membuat API web dengan titik akhir HTTP untuk Fungsi Peristiwa Lambda menggunakan Amazon API Gateway. API Gateway menyediakan alat untuk membuat dan mendokumentasikan API web yang merutekan permintaan HTTP ke Fungsi Peristiwa Lambda. Anda dapat mengamankan akses ke API Anda dengan kontrol autentikasi dan otorisasi. API Anda dapat melayani lalu lintas melalui internet atau hanya dapat diakses di dalam VPC Anda.

> **Tip**
>
> Lambda menawarkan dua cara untuk menjalankan fungsi Anda melalui titik akhir HTTP: API Gateway dan URL fungsi Lambda. Jika Anda tidak yakin metode mana yang terbaik untuk kasus penggunaan Anda, lihatthe section called “API Gateway vs URL fungsi”.

Sumber daya di API Anda menentukan satu atau beberapa metode, seperti GET atau POST. Metode memiliki integrasi yang merutekan permintaan ke Fungsi Peristiwa Lambda atau jenis integrasi lainnya. Anda dapat menentukan setiap sumber daya dan metode secara individual, atau menggunakan jenis sumber daya dan metode khusus untuk mencocokkan semua permintaan yang sesuai dengan suatu pola. Sumber daya proksi menangkap semua jalur di bawah sumber daya. Metode ANY menangkap semua metode HTTP.

Bagian-bagian

- Memilih jenis API
- Menambahkan titik akhir ke Fungsi Acara Lambda Anda
- Integrasi proxy
- Format peristiwa
- Format respons
- Izin
- Aplikasi sampel
- Penangan acara dari Powertools untuk AWS Lambda
- Tutorial: Menggunakan Lambda dengan API Gateway
- Menangani kesalahan Lambda dengan API Gateway API

- Pilih metode untuk memanggil Fungsi Peristiwa Lambda Anda menggunakan permintaan HTTP

Memilih jenis API

API Gateway mendukung tiga jenis API yang menjalankan Fungsi Peristiwa Lambda:

- HTTP API: RESTful API yang ringan dan latensi rendah.
- REST API: RESTful API yang dapat disesuaikan dan kaya fitur.
- WebSocket API: API web yang memelihara koneksi persisten dengan klien untuk komunikasi dupleks penuh.

API HTTP dan REST API adalah API RESTful yang memproses permintaan HTTP dan mengembalikan respons. API HTTP lebih baru dan dibangun dengan API dari API Gateway versi 2. Fitur berikut adalah fitur baru untuk API HTTP:

Fitur HTTP API

- Deployment otomatis – Saat Anda memodifikasi rute atau integrasi, mengubah deployment secara otomatis ke tahap yang deployment otomatisnya diaktifkan.
- Tahap default – Anda dapat membuat tahap default (`$default`) untuk melayani permintaan di jalur root URL API Anda. Untuk tahap yang diberi nama, Anda harus menyertakan nama tahap di awal jalur.
- Konfigurasi CORS – Anda dapat mengonfigurasi API Anda untuk menambahkan header CORS ke respons keluar, alih-alih menambahkannya secara manual dalam kode fungsi Anda.

REST API adalah API RESTful yang didukung oleh API Gateway sejak peluncuran. REST API saat ini memiliki lebih banyak kustomisasi, integrasi, dan fitur manajemen.

Fitur REST API

- Tipe integrasi – REST API mendukung integrasi Lambda kustom. Dengan integrasi kustom, Anda dapat mengirim hanya badan permintaan ke fungsi, atau menerapkan templat transformasi ke badan permintaan sebelum mengirimkannya ke fungsi.
- Kontrol akses – REST API mendukung lebih banyak opsi untuk autentikasi dan otorisasi.
- Pemantauan dan penelusuran — REST API mendukung opsi AWS X-Ray penelusuran dan pencatatan tambahan.

Untuk perbandingan mendetail, lihat Memilih antara API HTTP dan REST API di Panduan Pengembang API Gateway.

WebSocket API juga menggunakan API Gateway API versi 2 dan mendukung set fitur serupa. Gunakan WebSocket API untuk aplikasi yang mendapat manfaat dari koneksi persisten antara klien dan API. WebSocket API menyediakan komunikasi dupleks penuh, yang berarti bahwa klien dan API dapat mengirim pesan terus menerus tanpa menunggu respons.

API HTTP mendukung format peristiwa yang disederhanakan (versi 2.0). Untuk contoh peristiwa dari API HTTP, lihat Membuat integrasi AWS Lambda proxy untuk API HTTP di API Gateway.

Untuk informasi selengkapnya, lihat Membuat integrasi AWS Lambda proxy untuk API HTTP di API Gateway.

Menambahkan titik akhir ke Fungsi Acara Lambda Anda

Untuk menambahkan titik akhir publik ke Fungsi Acara Lambda Anda

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih fungsi.
3. Di bagian Gambaran umum fungsi, pilih Tambah pemicu.
4. Pilih API Gateway.
5. Pilih Buat API atau Gunakan API yang ada.
    a. API Baru: Untuk jenis API, pilih HTTP API. Untuk informasi selengkapnya, lihat Memilih jenis API.
    b. API yang ada: Pilih API dari daftar dropdown atau masukkan ID API (misalnya, r3pmxmplak).
6. Untuk Keamanan, pilih Terbuka.
7. Pilih Tambahkan.

Integrasi proxy

API dari API Gateway terdiri atas tahap, sumber daya, metode, dan integrasi. Tahap dan sumber daya menentukan jalur titik akhir:

Format jalur API

- `/prod/` – Tahap `prod` dan sumber daya root.

- `/prod/user` – Tahap `prod` dan sumber daya `user`.
- `/dev/{proxy+}` – Rute mana pun di tahap `dev`.
- `/` – (API HTTP) Tahap default dan sumber daya akar.

Integrasi Lambda memetakan jalur dan kombinasi metode HTTP ke Fungsi Peristiwa Lambda. Anda dapat mengonfigurasi API Gateway untuk mengirimkan badan permintaan HTTP sebagaimana adanya (integrasi kustom), atau untuk merangkum badan permintaan dalam dokumen yang mencakup semua informasi permintaan termasuk header, sumber daya, jalur, dan metode.

Untuk informasi selengkapnya, lihat Integrasi proxy Lambda di API Gateway.

Format peristiwa

Amazon API Gateway memanggil fungsi Anda secara sinkron dengan kejadian yang berisi representasi JSON permintaan HTTP. Untuk integrasi kustom, kejadian adalah badan dari permintaan. Untuk integrasi proksi, kejadian memiliki struktur yang telah ditentukan. Untuk contoh peristiwa proxy dari API REST API Gateway API, lihat Format input fungsi Lambda untuk integrasi proxy di Panduan Pengembang API Gateway.

Format respons

API Gateway menunggu respons dari fungsi Anda dan menyampaikan hasilnya kepada pemanggil. Untuk integrasi kustom, Anda menentukan respons integrasi dan respons metode untuk mengonversi output dari fungsi menjadi respons HTTP. Untuk integrasi proksi, fungsi harus merespons dengan representasi respons dalam format tertentu.

Contoh berikut menunjukkan objek respon dari Node.js fungsi. Objek respons mewakili respons HTTP yang berhasil yang berisi dokumen JSON.

Example index.mjs - Objek respons integrasi proxy () Node.js

```
var response = {
      "statusCode": 200,
      "headers": {
        "Content-Type": "application/json"
      },
      "isBase64Encoded": false,
      "multiValueHeaders": {
        "X-Custom-Header": ["My value", "My other value"],
      },
```

```
    "body": "{\n  \"TotalCodeSize\": 104330022,\n  \"FunctionCount\": 26\n}"
  }
```

Runtime Lambda menyusun objek respons menjadi JSON dan mengirimkannya ke API. API menguraikan respons dan menggunakannya untuk membuat respons HTTP, yang kemudian dikirimkan ke klien yang membuat permintaan asal.

Example Respons HTTP

```
< HTTP/1.1 200 OK
  < Content-Type: application/json
  < Content-Length: 55
  < Connection: keep-alive
  < x-amzn-RequestId: 32998fea-xmpl-4268-8c72-16138d629356
  < X-Custom-Header: My value
  < X-Custom-Header: My other value
  < X-Amzn-Trace-Id: Root=1-5e6aa925-ccecxmplbae116148e52f036
  <
  {
    "TotalCodeSize": 104330022,
    "FunctionCount": 26
  }
```

Izin

Amazon API Gateway mendapatkan izin untuk memanggil fungsi Anda dari kebijakan berbasis sumber daya milik fungsi tersebut. Anda dapat memberikan izin invokasi kepada seluruh API, atau memberikan akses terbatas kepada suatu tahap, sumber daya, atau metode.

Saat Anda menambahkan API ke fungsi Anda dengan menggunakan konsol Lambda, menggunakan konsol API Gateway, atau dalam templat AWS SAM , kebijakan berbasis sumber daya milik fungsi tersebut akan diperbarui secara otomatis. Berikut ini adalah contoh kebijakan fungsi.

Example kebijakan fungsi

JSON

```
{
  "Version":"2012-10-17",
  "Id": "default",
  "Statement": [
```

```
    {
      "Sid": "nodejs-apig-functiongetEndpointPermissionProd-BWDBXMPLXE2F",
      "Effect": "Allow",
      "Principal": {
        "Service": "apigateway.amazonaws.com"
      },
      "Action": "lambda:InvokeFunction",
      "Resource": "arn:aws:lambda:us-east-2:111122223333:function:nodejs-apig-
function-1G3MXMPLXVXYI",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "111122223333"
        },
        "ArnLike": {
          "aws:SourceArn": "arn:aws:execute-api:us-
east-2:111122223333:ktyvxmpls1/*/GET/"
        }
      }
    }
  ]
}
```

Anda dapat mengelola izin kebijakan fungsi secara manual dengan operasi API berikut:

- AddPermission
- RemovePermission
- GetPolicy

Untuk memberikan izin invokasi kepada API yang sudah ada, gunakan perintah `add-permission`. Contoh:

```
aws lambda add-permission \
  --function-name my-function \
  --statement-id apigateway-get --action lambda:InvokeFunction \
  --principal apigateway.amazonaws.com \
  --source-arn "arn:aws:execute-api:us-east-2:123456789012:mnh1xmpli7/default/GET/"
```

Anda akan melihat output berikut:

```
{
```

```
    "Statement": "{\"Sid\":\"apigateway-test-2\",\"Effect\":\"Allow\",\"Principal
\":{\"Service\":\"apigateway.amazonaws.com\"},\"Action\":\"lambda:InvokeFunction
\",\"Resource\":\"arn:aws:lambda:us-east-2:123456789012:function:my-function
\",\"Condition\":{\"ArnLike\":{\"AWS:SourceArn\":\"arn:aws:execute-api:us-
east-2:123456789012:mnh1xmpli7/default/GET\"}}}"
}
```

> **Note**
>
> Jika fungsi dan API Anda berbeda Wilayah AWS, pengenal Region di ARN sumber harus cocok dengan Region fungsi, bukan Region API. Saat API Gateway memanggil fungsi, ia menggunakan ARN sumber daya yang didasarkan pada ARN API, tetapi dimodifikasi agar sesuai dengan Wilayah fungsi.

ARN sumber dalam contoh ini memberikan izin ke integrasi di metode GET dari sumber daya akar di tahap default API, dengan `mnh1xmpli7` ID. Anda dapat menggunakan tanda bintang di ARN sumber untuk memberikan izin untuk beberapa tahap, metode, atau sumber daya.

Pola sumber daya

- `mnh1xmpli7/*/GET/*` – Metode GET di semua sumber daya di semua tahap.
- `mnh1xmpli7/prod/ANY/user` – Metode ANY pada sumber daya `user` di tahap `prod`.
- `mnh1xmpli7/*/*/*` – Metode apa pun pada semua sumber daya di semua tahap.

Untuk perincian tentang melihat kebijakan dan menghapus pernyataan, lihat Bekerja dengan kebijakan berbasis sumber daya di Lambda.

Aplikasi sampel

API Gateway dengan aplikasi Node.js sampel menyertakan fungsi dengan AWS SAM templat yang membuat REST API yang mengaktifkan AWS X-Ray penelusuran. Ini juga mencakup skrip untuk menerapkan, menjalankan fungsi, menguji API, dan pembersihan.

Penangan acara dari Powertools untuk AWS Lambda

Event handler dari Powertools for AWS Lambda toolkit menyediakan routing, middleware, konfigurasi CORS, pembuatan spesifikasi OpenAPI, validasi permintaan, penanganan kesalahan, dan fitur berguna lainnya saat menulis Lambda Event Functions yang dipanggil oleh titik akhir API Gateway

(HTTP atau REST). Utilitas event handler tersedia untuk TypeScript/JavaScript Python dan. Untuk informasi selengkapnya, lihat Event Handler REST API dalam dokumentasi Powertools for AWS Lambda (Python) dan Event Handler HTTP API dalam dokumentasi Powertools for Lambda (). AWS TypeScript

Python

```
from aws_lambda_powertools import Logger
from aws_lambda_powertools.event_handler import APIGatewayRestResolver
from aws_lambda_powertools.logging import correlation_paths
from aws_lambda_powertools.utilities.typing.lambda_context import LambdaContext

app = APIGatewayRestResolver()
logger = Logger()

@app.get("/healthz")
def ping():
    return {"message": "health status ok"}

@logger.inject_lambda_context(correlation_id_path=correlation_paths.API_GATEWAY_REST)
def lambda_handler(event: dict, context: LambdaContext) -> dict:
    return app.resolve(event, context)
```

TypeScript

```
import { Router } from '@aws-lambda-powertools/event-handler/experimental-rest';
import { Logger } from '@aws-lambda-powertools/logger';
import {
  correlationPaths,
  search,
} from '@aws-lambda-powertools/logger/correlationId';
import type { Context } from 'aws-lambda/handler';

const logger = new Logger({
  correlationIdSearchFn: search,
});

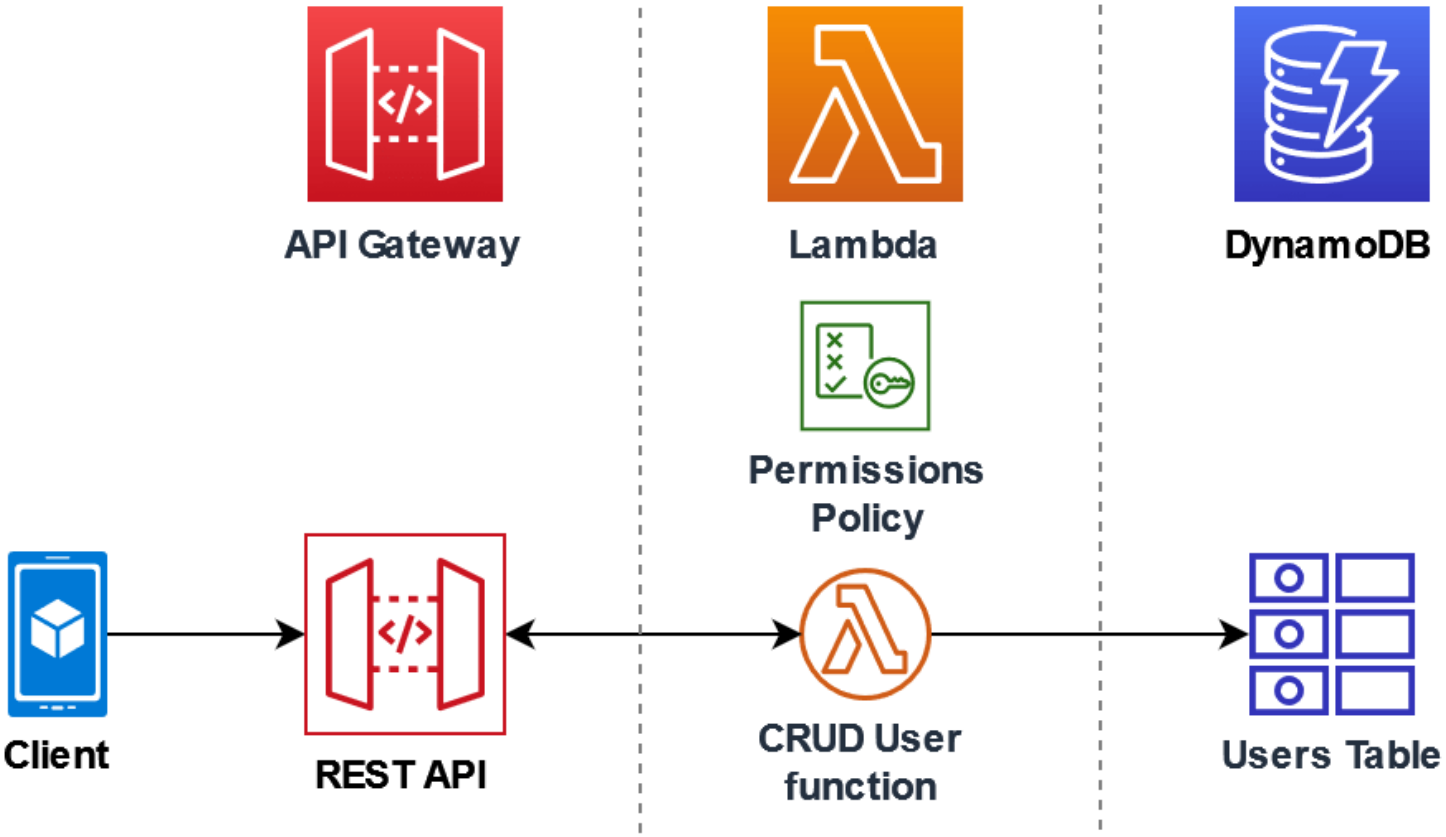
const app = new Router({ logger });

app.get("/healthz", async () => {
  return { message: "health status ok" };
});
```

```
export const handler = async (event: unknown, context: Context) => {
  // You can continue using other utilities just as before
  logger.addContext(context);
  logger.setCorrelationId(event, correlationPaths.API_GATEWAY_REST);
  return app.resolve(event, context);
};
```

Tutorial: Menggunakan Lambda dengan API Gateway

Dalam tutorial ini, Anda membuat REST API yang melaluinya Anda menjalankan Fungsi Peristiwa Lambda menggunakan permintaan HTTP. Fungsi Acara Lambda Anda melakukan operasi membuat, membaca, memperbarui, dan menghapus (CRUD) pada tabel DynamoDB. Fungsi ini disediakan di sini untuk demonstrasi, tetapi Anda belajar mengonfigurasi API REST API Gateway yang dapat memanggil Fungsi Peristiwa Lambda apa pun.



Menggunakan API Gateway memberi pengguna titik akhir HTTP yang aman untuk menjalankan Fungsi Peristiwa Lambda Anda dan dapat membantu mengelola panggilan dalam jumlah besar ke fungsi Anda dengan membatasi lalu lintas dan secara otomatis memvalidasi dan mengotorisasi panggilan API. API Gateway juga menyediakan kontrol keamanan fleksibel menggunakan AWS

Identity and Access Management (IAM) dan Amazon Cognito. Ini berguna untuk kasus penggunaan di mana otorisasi terlebih dahulu diperlukan untuk panggilan ke aplikasi Anda.

> **Tip**
>
> Lambda menawarkan dua cara untuk menjalankan fungsi Anda melalui titik akhir HTTP: API Gateway dan URL fungsi Lambda. Jika Anda tidak yakin metode mana yang terbaik untuk kasus penggunaan Anda, lihatthe section called “API Gateway vs URL fungsi”.

Untuk menyelesaikan tutorial ini, Anda melalui tahapan berikut:

1. Membuat dan mengkonfigurasi Lambda Event Function di Python atau Node.js untuk melakukan operasi pada tabel DynamoDB.
2. Buat REST API di API Gateway untuk terhubung ke Fungsi Peristiwa Lambda Anda.
3. Buat tabel DynamoDB dan mengujinya dengan Lambda Event Function Anda di konsol.
4. Terapkan API Anda dan uji penyiapan lengkap menggunakan curl di terminal.

Dengan menyelesaikan tahapan ini, Anda mempelajari cara menggunakan API Gateway untuk membuat titik akhir HTTP yang dapat menjalankan Fungsi Peristiwa Lambda dengan aman pada skala apa pun. Anda juga mempelajari cara menerapkan API Anda, dan cara mengujinya di konsol dan dengan mengirimkan permintaan HTTP menggunakan terminal.

Membuat kebijakan izin

Sebelum Anda dapat membuat peran eksekusi untuk Fungsi Peristiwa Lambda, Anda harus terlebih dahulu membuat kebijakan izin untuk memberikan izin fungsi Anda untuk mengakses sumber daya yang diperlukan. AWS Untuk tutorial ini, kebijakan ini memungkinkan Lambda untuk melakukan operasi CRUD pada tabel DynamoDB dan menulis ke Amazon Logs. CloudWatch

Untuk membuat kebijakan

1. Buka halaman Kebijakan konsol IAM.
2. Pilih Buat Kebijakan.
3. Pilih tab JSON, lalu tempelkan kebijakan khusus berikut ke editor JSON.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt1428341300017",
      "Action": [
        "dynamodb:DeleteItem",
        "dynamodb:GetItem",
        "dynamodb:PutItem",
        "dynamodb:Query",
        "dynamodb:Scan",
        "dynamodb:UpdateItem"
      ],
      "Effect": "Allow",
      "Resource": "*"
    },
    {
      "Sid": "",
      "Resource": "*",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Effect": "Allow"
    }
  ]
}
```

4. Pilih Berikutnya: Tanda.
5. Pilih Berikutnya: Tinjau.
6. Di bawah Kebijakan peninjauan, untuk Nama kebijakan, masukkan**`lambda-apigateway-policy`**.
7. Pilih Buat kebijakan.

Membuat peran eksekusi

Peran eksekusi adalah peran AWS Identity and Access Management (IAM) yang memberikan izin Fungsi Peristiwa Lambda untuk mengakses dan sumber daya. Layanan AWS Untuk mengaktifkan fungsi Anda untuk melakukan operasi pada tabel DynamoDB, Anda melampirkan kebijakan izin yang Anda buat di langkah sebelumnya.

Untuk membuat peran eksekusi dan melampirkan kebijakan izin kustom

1. Buka halaman Peran dari konsol IAM.
2. Pilih Buat peran.
3. Untuk jenis entitas tepercaya, pilih AWS layanan, lalu untuk kasus penggunaan, pilih Lambda.
4. Pilih Berikutnya.
5. Dalam kotak pencarian kebijakan, masukkan **`lambda-apigateway-policy`**.
6. Di hasil penelusuran, pilih kebijakan yang Anda buat (`lambda-apigateway-policy`), lalu pilih Berikutnya.
7. Di bawah Rincian peran, untuk nama Peran, masukkan**`lambda-apigateway-role`**, lalu pilih Buat peran.

Buat Fungsi Acara Lambda

1. Buka halaman Fungsi Acara konsol Lambda dan pilih Buat Fungsi Acara.
2. Pilih Tulis dari awal.
3. Untuk Nama fungsi, masukkan `LambdaFunctionOverHttps`.
4. Untuk Runtime, pilih runtime terbaru Node.js atau Python.
5. (Opsional) Di bagian Izin, luaskan Ubah peran eksekusi default.
6. Pilih Gunakan peran yang ada, lalu pilih **`lambda-apigateway-role`** peran yang Anda buat sebelumnya.
7. Pilih Buat Fungsi Acara.
8. Di panel sumber Kode, ganti kode default dengan kode berikut Node.js atau Python.

   Node.js

   `region`Pengaturan harus sesuai dengan Wilayah AWS tempat Anda menyebarkan fungsi dan membuat tabel DynamoDB.

Example index.mjs

```
import { DynamoDBDocumentClient, PutCommand, GetCommand,
         UpdateCommand, DeleteCommand} from "@aws-sdk/lib-dynamodb";
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

const ddbClient = new DynamoDBClient({ region: "us-east-2" });
const ddbDocClient = DynamoDBDocumentClient.from(ddbClient);

// Define the name of the DDB table to perform the CRUD operations on
const tablename = "lambda-apigateway";

/**
 * Provide an event that contains the following keys:
 *
 *   - operation: one of 'create,' 'read,' 'update,' 'delete,' or 'echo'
 *   - payload: a JSON object containing the parameters for the table item
 *     to perform the operation on
 */
export const handler = async (event, context) => {

     const operation = event.operation;

     if (operation == 'echo'){
          return(event.payload);
     }

    else {
        event.payload.TableName = tablename;
        let response;

        switch (operation) {
          case 'create':
               response = await ddbDocClient.send(new
 PutCommand(event.payload));
               break;
          case 'read':
               response = await ddbDocClient.send(new
 GetCommand(event.payload));
               break;
          case 'update':
               response = ddbDocClient.send(new UpdateCommand(event.payload));
               break;
```

```
          case 'delete':
               response = ddbDocClient.send(new DeleteCommand(event.payload));
               break;
          default:
            response = 'Unknown operation: ${operation}';
          }
        console.log(response);
        return response;
    }
};
```

Python

Example lambda_function.py

```
import boto3

# Define the DynamoDB table that Lambda will connect to
table_name = "lambda-apigateway"

# Create the DynamoDB resource
dynamo = boto3.resource('dynamodb').Table(table_name)

# Define some functions to perform the CRUD operations
def create(payload):
    return dynamo.put_item(Item=payload['Item'])

def read(payload):
    return dynamo.get_item(Key=payload['Key'])

def update(payload):
    return dynamo.update_item(**{k: payload[k] for k in ['Key',
 'UpdateExpression',
    'ExpressionAttributeNames', 'ExpressionAttributeValues'] if k in payload})

def delete(payload):
    return dynamo.delete_item(Key=payload['Key'])

def echo(payload):
    return payload

operations = {
    'create': create,
```

```
    'read': read,
    'update': update,
    'delete': delete,
    'echo': echo,
}

def lambda_handler(event, context):
    '''Provide an event that contains the following keys:
      - operation: one of the operations in the operations dict below
      - payload: a JSON object containing parameters to pass to the
        operation being performed
    '''

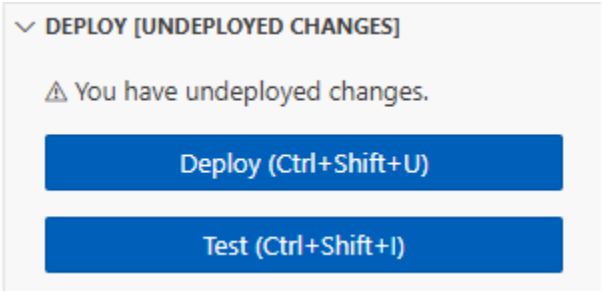
    operation = event['operation']
    payload = event['payload']

    if operation in operations:
        return operations[operation](payload)

    else:
        raise ValueError(f'Unrecognized operation "{operation}"')
```

> **Note**
>
> Dalam contoh ini, nama tabel DynamoDB didefinisikan sebagai variabel dalam kode fungsi Anda. Dalam aplikasi nyata, praktik terbaik adalah meneruskan parameter ini sebagai variabel lingkungan dan untuk menghindari hardcoding nama tabel. Untuk informasi selengkapnya lihat Menggunakan variabel AWS Lambda lingkungan.

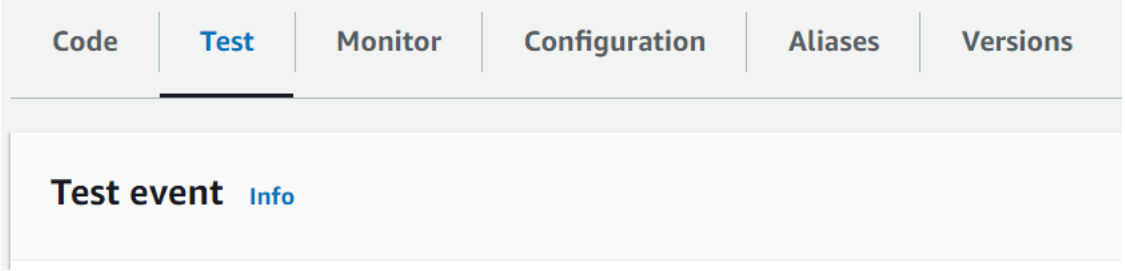
9. Di bagian DEPLOY, pilih Deploy untuk memperbarui kode fungsi Anda:



Uji Fungsi Acara

Sebelum mengintegrasikan fungsi Anda dengan API Gateway, konfirmasikan bahwa Anda telah berhasil menerapkan fungsi tersebut. Gunakan konsol Lambda untuk mengirim acara pengujian ke fungsi Anda.

1. Pada halaman konsol Lambda untuk fungsi Anda, pilih tab Uji.



2. Gulir ke bawah ke bagian Event JSON dan ganti acara default dengan yang berikut ini. Acara ini cocok dengan struktur yang diharapkan oleh Lambda Event Function.

```
{
    "operation": "echo",
    "payload": {
        "somekey1": "somevalue1",
        "somekey2": "somevalue2"
    }
}
```

3. Pilih Uji.
4. Di bawah Fungsi Eksekusi: berhasil, perluas Detail. Anda akan melihat tanggapan berikut:

```
{
  "somekey1": "somevalue1",
  "somekey2": "somevalue2"
}
```

Membuat REST API menggunakan API Gateway

Pada langkah ini, Anda membuat API Gateway REST API yang Anda gunakan untuk menjalankan Fungsi Peristiwa Lambda Anda.

Untuk membuat API

1. Buka konsol API Gateway.
2. Pilih Buat API.
3. Di kotak REST API, pilih Build.
4. Di bawah detail API, biarkan API Baru dipilih, dan untuk Nama API, masukkan**`DynamoDBOperations`**.
5. Pilih Buat API.

Buat sumber daya di REST API

Untuk menambahkan metode HTTP ke API Anda, Anda harus terlebih dahulu membuat sumber daya agar metode tersebut dapat dioperasikan. Di sini Anda membuat sumber daya untuk mengelola tabel DynamoDB Anda.

Untuk membuat sumber daya

1. Di konsol API Gateway, pada halaman Resources untuk API Anda, pilih Buat sumber daya.
2. Dalam Rincian sumber daya, untuk nama Sumber daya masukkan**`DynamoDBManager`**.
3. Pilih Buat Sumber Daya.

Buat metode HTTP POST

Pada langkah ini, Anda membuat metode (`POST`) untuk `DynamoDBManager` sumber daya Anda. Anda menautkan `POST` metode ini ke Fungsi Peristiwa Lambda sehingga ketika metode menerima permintaan HTTP, API Gateway memanggil Fungsi Peristiwa Lambda Anda.

> **Note**
>
> Untuk tujuan tutorial ini, satu metode HTTP (`POST`) digunakan untuk memanggil Lambda Event Function tunggal yang melakukan semua operasi pada tabel DynamoDB Anda. Dalam aplikasi nyata, praktik terbaik adalah menggunakan Fungsi Peristiwa Lambda dan metode HTTP yang berbeda untuk setiap operasi. Untuk informasi lebih lanjut, lihat Monolit Lambda di Tanah Tanpa Server.

Untuk membuat metode POST

1. Pada halaman Resources untuk API Anda, pastikan `/DynamoDBManager` sumber daya disorot. Kemudian, di panel Methods, pilih Create method.
2. Untuk jenis Metode, pilih POST.
3. Untuk jenis Integrasi, biarkan fungsi Lambda dipilih.
4. Untuk fungsi Lambda, pilih Amazon Resource Name (ARN) untuk function (). `LambdaFunctionOverHttps`
5. Pilih metode Buat.

Membuat tabel DynamoDB

Buat tabel DynamoDB kosong tempat Fungsi Peristiwa Lambda Anda melakukan operasi CRUD.

Untuk membuat tabel DynamoDB

1. Buka halaman Tabel di konsol DynamoDB.
2. Pilih Buat tabel.
3. Di bawah Rincian tabel, lakukan hal berikut:
    1. Untuk Nama tabel, masukkan **`lambda-apigateway`**.
    2. Untuk kunci Partition, masukkan**`id`**, dan simpan tipe data yang ditetapkan sebagai String.
4. Di bawah Pengaturan tabel, pertahankan pengaturan Default.
5. Pilih Buat tabel.

Uji integrasi API Gateway, Lambda, dan DynamoDB

Anda sekarang siap untuk menguji integrasi metode API Gateway API Anda dengan Lambda Event Function dan tabel DynamoDB Anda. Menggunakan konsol API Gateway, Anda mengirim permintaan langsung ke `POST` metode menggunakan fungsi pengujian konsol. Pada langkah ini, Anda pertama kali menggunakan `create` operasi untuk menambahkan item baru ke tabel DynamoDB Anda, kemudian Anda menggunakan operasi untuk `update` memodifikasi item.

Tes 1: Untuk membuat item baru di tabel DynamoDB

1. Di konsol API Gateway, pilih API (`DynamoDBOperations`) Anda.
2. Pilih metode POST di bawah `DynamoDBManager` sumber daya.

3. Pilih tab Uji. Anda mungkin perlu memilih tombol panah kanan untuk menampilkan tab.
4. Di bawah metode Test, biarkan string Query dan Header kosong. Untuk badan Permintaan, tempel JSON berikut:

```
{
  "operation": "create",
  "payload": {
    "Item": {
      "id": "1234ABCD",
      "number": 5
    }
  }
}
```

5. Pilih Uji.

   Hasil yang ditampilkan saat tes selesai harus menunjukkan status`200`. Kode status ini menunjukkan bahwa `create` operasi berhasil.

   Untuk mengonfirmasi, periksa apakah tabel DynamoDB Anda sekarang berisi item baru.

6. Buka halaman Tabel konsol DynamoDB dan pilih tabel. `lambda-apigateway`
7. Pilih Jelajahi item tabel. Di panel Item yang dikembalikan, Anda akan melihat satu item dengan id `1234ABCD` dan nomornya`5`. Contoh:



Tes 2: Untuk memperbarui item di tabel DynamoDB Anda

1. Di konsol API Gateway, kembali ke tab Test metode POST Anda.
2. Di bawah metode Test, biarkan string Query dan Header kosong. Untuk badan Permintaan, tempel JSON berikut:

```
{
```

```
    "operation": "update",
    "payload": {
        "Key": {
            "id": "1234ABCD"
        },
        "UpdateExpression": "SET #num = :newNum",
        "ExpressionAttributeNames": {
            "#num": "number"
        },
        "ExpressionAttributeValues": {
            ":newNum": 10
        }
    }
}
```

3. Pilih Uji.

   Hasil yang ditampilkan saat tes selesai harus menunjukkan status`200`. Kode status ini menunjukkan bahwa `update` operasi berhasil.

   Untuk mengonfirmasi, periksa apakah item di tabel DynamoDB Anda telah dimodifikasi.

4. Buka halaman Tabel konsol DynamoDB dan pilih tabel. `lambda-apigateway`
5. Pilih Jelajahi item tabel. Di panel Item yang dikembalikan, Anda akan melihat satu item dengan id `1234ABCD` dan nomornya`10`.



Terapkan API

Agar klien dapat memanggil API, Anda harus membuat penerapan dan tahap terkait. Tahap mewakili snapshot API Anda termasuk metode dan integrasinya.

Untuk menerapkan API

1. Buka halaman API dari konsol API Gateway dan pilih `DynamoDBOperations` API.

2. Pada halaman Sumber Daya untuk API Anda, pilih Deploy API.
3. Untuk Stage, pilih *Tahap baru*, lalu untuk nama Stage, masukkan. **`test`**
4. Pilih Deploy.
5. Di panel Detail tahap, salin URL Panggilan. Anda menggunakan ini di langkah berikutnya untuk memanggil fungsi Anda menggunakan permintaan HTTP.

Gunakan curl untuk memanggil Fungsi Acara Anda menggunakan permintaan HTTP

Anda sekarang dapat memanggil Lambda Event Function Anda dengan mengeluarkan permintaan HTTP ke API Anda. Pada langkah ini, Anda membuat item baru di tabel DynamoDB Anda dan kemudian melakukan operasi baca, perbarui, dan hapus pada item tersebut.

Untuk membuat item di tabel DynamoDB Anda menggunakan curl

1. Buka terminal atau command prompt pada mesin lokal Anda dan jalankan `curl` perintah berikut menggunakan URL pemanggilan yang Anda salin pada langkah sebelumnya. Perintah ini menggunakan opsi berikut:

   - `-H`: Menambahkan header kustom untuk permintaan. Di sini, ia menentukan jenis konten sebagai JSON.
   - `-d`: Mengirim data di badan permintaan. Opsi ini menggunakan metode HTTP POST secara default.

   Linux/macOS

   ```
   curl https://l8togsqxd8.execute-api.us-east-2.amazonaws.com/test/DynamoDBManager
    \
   -H "Content-Type: application/json" \
   -d '{"operation": "create", "payload": {"Item": {"id": "5678EFGH", "number":
    15}}}'
   ```

   PowerShell

   ```
   curl.exe 'https://l8togsqxd8.execute-api.us-east-2.amazonaws.com/test/
   DynamoDBManager' -H 'Content-Type: application/json' -d '{\"operation\":
    \"create\", \"payload\": {\"Item\": {\"id\": \"5678EFGH\", \"number\": 15}}}'
   ```

Jika operasi berhasil, Anda akan melihat respons yang dikembalikan dengan kode status HTTP 200.

2. Anda juga dapat menggunakan konsol DynamoDB untuk memverifikasi bahwa item baru ada di tabel Anda dengan melakukan hal berikut:

   1. Buka halaman Tabel konsol DynamoDB dan pilih tabel. `lambda-apigateway`
   2. Pilih Jelajahi item tabel. Di panel Item yang dikembalikan, Anda akan melihat item dengan id `5678EFGH` dan nomornya`15`.

Untuk membaca item di tabel DynamoDB Anda menggunakan curl

- Di terminal atau command prompt Anda, jalankan `curl` perintah berikut untuk membaca nilai item yang baru saja Anda buat. Gunakan URL pemanggilan Anda sendiri.

  Linux/macOS

  ```
  curl https://l8togsqxd8.execute-api.us-east-2.amazonaws.com/test/DynamoDBManager
   \
  -H "Content-Type: application/json" \
  -d '{"operation": "read", "payload": {"Key": {"id": "5678EFGH"}}}'
  ```

  PowerShell

  ```
  curl.exe 'https://l8togsqxd8.execute-api.us-east-2.amazonaws.com/test/
  DynamoDBManager' -H 'Content-Type: application/json' -d '{\"operation\": \"read
  \", \"payload\": {\"Key\": {\"id\": \"5678EFGH\"}}}'
  ```

  Anda akan melihat output seperti salah satu dari berikut ini tergantung pada apakah Anda memilih Node.js atau kode fungsi Python:

  Node.js

  ```
  {"$metadata":
  {"httpStatusCode":200,"requestId":"7BP3G5Q0C0O1E50FBQI9NS099JVV4KQNSO5AEMVJF66Q9ASUAAJG"
  "attempts":1,"totalRetryDelay":0},"Item":{"id":"5678EFGH","number":15}}
  ```

Python

```
{"Item":{"id":"5678EFGH","number":15},"ResponseMetadata":
{"RequestId":"QNDJICE52E86B82VETR6RKBE5BVV4KQNSO5AEMVJF66Q9ASUAAJG",
"HTTPStatusCode":200,"HTTPHeaders":{"server":"Server","date":"Wed, 31 Jul 2024
 00:37:01 GMT","content-type":"application/x-amz-json-1.0",
"content-length":"52","connection":"keep-alive","x-amzn-
requestid":"QNDJICE52E86B82VETR6RKBE5BVV4KQNSO5AEMVJF66Q9ASUAAJG","x-amz-
crc32":"2589610852"},
"RetryAttempts":0}}
```

Untuk memperbarui item dalam tabel DynamoDB Anda menggunakan curl

1. Di terminal atau command prompt Anda, jalankan `curl` perintah berikut untuk memperbarui item yang baru saja Anda buat dengan mengubah `number` nilainya. Gunakan URL pemanggilan Anda sendiri.

   Linux/macOS

   ```
   curl https://l8togsqxd8.execute-api.us-east-2.amazonaws.com/test/DynamoDBManager
    \
   -H "Content-Type: application/json" \
   -d '{"operation": "update", "payload": {"Key": {"id": "5678EFGH"},
    "UpdateExpression": "SET #num = :new_value", "ExpressionAttributeNames":
    {"#num": "number"}, "ExpressionAttributeValues": {":new_value": 42}}}'
   ```

   PowerShell

   ```
   curl.exe 'https://l8togsqxd8.execute-api.us-east-2.amazonaws.com/test/
   DynamoDBManager' -H 'Content-Type: application/json' -d '{\"operation\":
    \"update\", \"payload\": {\"Key\": {\"id\": \"5678EFGH\"}, \"UpdateExpression
   \": \"SET #num = :new_value\", \"ExpressionAttributeNames\": {\"#num\": \"number
   \"}, \"ExpressionAttributeValues\": {\":new_value\": 42}}}'
   ```

2. Untuk mengonfirmasi bahwa nilai `number` item telah diperbarui, jalankan perintah baca lainnya:

   Linux/macOS

   ```
   curl https://l8togsqxd8.execute-api.us-east-2.amazonaws.com/test/DynamoDBManager
    \
   ```

```
-H "Content-Type: application/json" \
-d '{"operation": "read", "payload": {"Key": {"id": "5678EFGH"}}}'
```

PowerShell

```
curl.exe 'https://l8togsqxd8.execute-api.us-east-2.amazonaws.com/test/
DynamoDBManager' -H 'Content-Type: application/json' -d '{\"operation\": \"read
\", \"payload\": {\"Key\": {\"id\": \"5678EFGH\"}}}'
```

Untuk menghapus item dalam tabel DynamoDB Anda menggunakan curl

1. Di terminal atau command prompt Anda, jalankan `curl` perintah berikut untuk menghapus item yang baru saja Anda buat. Gunakan URL pemanggilan Anda sendiri.

   Linux/macOS

   ```
   curl https://l8togsqxd8.execute-api.us-east-2.amazonaws.com/test/DynamoDBManager
    \
   -H "Content-Type: application/json" \
   -d '{"operation": "delete", "payload": {"Key": {"id": "5678EFGH"}}}'
   ```

   PowerShell

   ```
   curl.exe 'https://l8togsqxd8.execute-api.us-east-2.amazonaws.com/test/
   DynamoDBManager' -H 'Content-Type: application/json' -d '{\"operation\":
    \"delete\", \"payload\": {\"Key\": {\"id\": \"5678EFGH\"}}}'
   ```

2. Konfirmasikan bahwa operasi penghapusan berhasil. Di panel Item yang dikembalikan dari halaman item Jelajahi konsol DynamoDB, verifikasi bahwa item dengan `5678EFGH` id tidak lagi ada di tabel.

Bersihkan sumber daya Anda (opsional)

Sekarang Anda dapat menghapus sumber daya yang Anda buat untuk tutorial ini, kecuali Anda ingin mempertahankannya. Dengan menghapus AWS sumber daya yang tidak lagi Anda gunakan, Anda mencegah tagihan yang tidak perlu ke Anda Akun AWS.

Untuk menghapus Fungsi Acara Lambda

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih Fungsi Acara yang Anda buat.
3. Pilih Tindakan, Hapus.
4. Ketik **`confirm`** kolom input teks dan pilih Hapus.

Untuk menghapus peran eksekusi

1. Buka halaman Peran dari konsol IAM.
2. Pilih peran eksekusi yang Anda buat.
3. Pilih Hapus.
4. Masukkan nama peran di bidang input teks dan pilih Hapus.

Untuk menghapus API

1. Buka halaman API di konsol API Gateway.
2. Pilih API yang Anda buat.
3. Pilih Tindakan, Hapus.
4. Pilih Hapus.

Untuk menghapus tabel DynamoDB

1. Buka halaman Tabel di konsol DynamoDB.
2. Pilih tabel yang Anda buat.
3. Pilih Hapus.
4. Masukkan **`delete`** di kotak teks.
5. Pilih Hapus tabel.

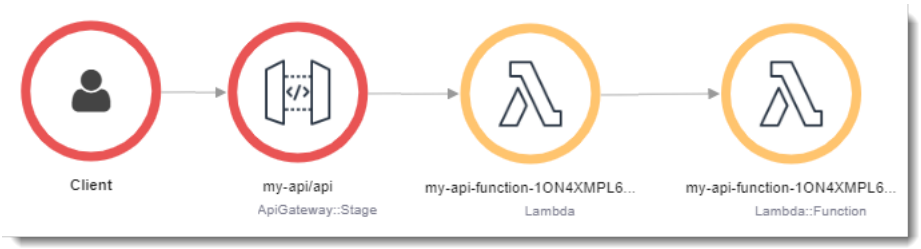
Jika Anda menemukan kesalahan saat menjalankan fungsi Anda melalui API Gateway, lihatthe section called “Kesalahan”.

Menangani kesalahan Lambda dengan API Gateway API

API Gateway memperlakukan semua kesalahan pemanggilan dan Fungsi Peristiwa sebagai kesalahan internal. Jika API Lambda menolak permintaan invokasi, API Gateway mengembalikan kode kesalahan 500. Jika Fungsi Peristiwa berjalan tetapi mengembalikan kesalahan, atau mengembalikan respons dalam format yang salah, API Gateway mengembalikan 502. Dalam kedua kasus tersebut, badan respons dari API Gateway adalah `{"message": "Internal server error"}`.

> **Note**
>
> API Gateway tidak mencoba lagi invokasi Lambda apa pun. Jika Lambda mengembalikan kesalahan, API Gateway mengembalikan respons kesalahan ke klien.

Contoh berikut menunjukkan peta X-Ray jejak untuk permintaan yang mengakibatkan kesalahan fungsi dan 502 dari API Gateway. Klien menerima pesan kesalahan umum.



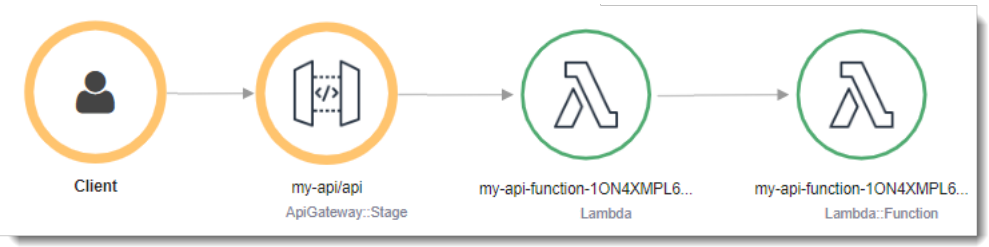
Untuk menyesuaikan respons kesalahan, Anda harus menangkap kesalahan dalam kode dan memformat tanggapan dalam format yang diperlukan.

Example index.mjs — Kesalahan pemformatan

```
var formatError = function(error){
  var response = {
    "statusCode": error.statusCode,
    "headers": {
      "Content-Type": "text/plain",
      "x-amzn-ErrorType": error.code
    },
    "isBase64Encoded": false,
    "body": error.code + ": " + error.message
  }
  return response
```

```
}
```

API Gateway mengonversi respons ini menjadi kesalahan HTTP dengan kode dan badan status kustom. Di peta jejak, node fungsi berwarna hijau karena menangani kesalahan tersebut.



Pilih metode untuk memanggil Fungsi Peristiwa Lambda Anda menggunakan permintaan HTTP

Banyak kasus penggunaan umum untuk Lambda melibatkan pemanggilan Fungsi Acara Anda menggunakan permintaan HTTP. Misalnya, Anda mungkin ingin aplikasi web menjalankan fungsi Anda melalui permintaan browser. Fungsi Peristiwa Lambda juga dapat digunakan untuk membuat API REST lengkap, menangani interaksi pengguna dari aplikasi seluler, memproses data dari layanan eksternal melalui panggilan HTTP, atau membuat webhook khusus.

Bagian berikut menjelaskan pilihan Anda untuk memanggil Lambda melalui HTTP dan memberikan informasi untuk membantu Anda membuat keputusan yang tepat untuk kasus penggunaan khusus Anda.

Apa pilihan Anda saat memilih metode pemanggilan HTTP?

Lambda menawarkan dua metode utama untuk menjalankan fungsi menggunakan permintaan HTTP - URL fungsi dan API Gateway. Perbedaan utama antara kedua opsi ini adalah sebagai berikut:

- URL fungsi Lambda menyediakan titik akhir HTTP langsung yang sederhana untuk Fungsi Peristiwa Lambda. Mereka dioptimalkan untuk kesederhanaan dan efektivitas biaya dan menyediakan jalur tercepat untuk mengekspos Fungsi Peristiwa Lambda melalui HTTP.
- API Gateway adalah layanan yang lebih canggih untuk membangun API berfitur lengkap. API Gateway dioptimalkan untuk membangun dan mengelola API produksi dalam skala besar dan menyediakan alat komprehensif untuk keamanan, pemantauan, dan manajemen lalu lintas.

Rekomendasi jika Anda sudah mengetahui kebutuhan Anda

Jika Anda sudah jelas tentang kebutuhan Anda, berikut adalah rekomendasi dasar kami:

Kami merekomendasikan URL fungsi untuk aplikasi sederhana atau prototipe di mana Anda hanya memerlukan metode otentikasi dasar dan request/response penanganan dan di mana Anda ingin meminimalkan biaya dan kompleksitas.

API Gateway adalah pilihan yang lebih baik untuk aplikasi produksi dalam skala besar atau untuk kasus di mana Anda memerlukan fitur yang lebih canggih seperti dukungan OpenAPI Description, pilihan opsi otentikasi, nama domain khusus, atau request/response penanganan kaya termasuk throttling, caching, dan transformasi. request/response

Apa yang harus dipertimbangkan saat memilih metode untuk menjalankan Fungsi Acara Lambda Anda

Saat memilih antara URL fungsi dan API Gateway, Anda perlu mempertimbangkan faktor-faktor berikut:

- Kebutuhan otentikasi Anda, seperti apakah Anda memerlukan OAuth atau Amazon Cognito untuk mengautentikasi pengguna
- Persyaratan penskalaan Anda dan kompleksitas API yang ingin Anda terapkan
- Apakah Anda memerlukan fitur canggih seperti validasi permintaan dan request/response pemformatan
- Persyaratan pemantauan Anda
- Sasaran biaya Anda

Dengan memahami faktor-faktor ini, Anda dapat memilih opsi yang paling menyeimbangkan persyaratan keamanan, kompleksitas, dan biaya Anda.

Informasi berikut merangkum perbedaan utama antara kedua opsi.

Autentikasi

- URL fungsi menyediakan opsi otentikasi dasar melalui AWS Identity and Access Management (IAM). Anda dapat mengonfigurasi titik akhir Anda menjadi publik (tidak ada otentikasi) atau memerlukan otentikasi IAM. Dengan autentikasi IAM, Anda dapat menggunakan AWS kredensyal standar atau peran IAM untuk mengontrol akses. Meskipun mudah diatur, pendekatan ini memberikan opsi terbatas dibandingkan dengan metode otentikat lainnya.
- API Gateway menyediakan akses ke berbagai opsi otentikasi yang lebih komprehensif. Selain autentikasi IAM, Anda dapat menggunakan otorisasi Lambda (logika otentikasi khusus), kumpulan

pengguna Amazon Cognito, dan alur. OAuth2.0 Fleksibilitas ini memungkinkan Anda menerapkan skema otentikasi yang kompleks, termasuk penyedia otentikasi pihak ketiga, otentikasi berbasis token, dan otentikasi multi-faktor.

Request/response penanganan

- URL fungsi menyediakan permintaan HTTP dasar dan penanganan respons. Mereka mendukung metode HTTP standar dan menyertakan dukungan berbagi sumber daya lintas asal (CORS) bawaan. Meskipun mereka dapat menangani muatan JSON dan parameter kueri secara alami, mereka tidak menawarkan transformasi permintaan atau kemampuan validasi. Penanganan respons sama mudahnya — klien menerima respons dari Fungsi Peristiwa Lambda Anda persis seperti Lambda mengembalikannya.
- API Gateway menyediakan kemampuan penanganan permintaan dan respons yang canggih. Anda dapat menentukan validator permintaan, mengubah permintaan dan tanggapan menggunakan templat pemetaan, menyiapkan request/response header, dan menerapkan cache respons. API Gateway juga mendukung payload biner dan nama domain khusus dan dapat memodifikasi tanggapan sebelum mencapai klien. Anda dapat mengatur model untuk request/response validasi dan transformasi menggunakan JSON Schema.

Penskalaan

- URL fungsi menskalakan secara langsung dengan batas konkurensi Fungsi Acara Lambda Anda dan menangani lonjakan lalu lintas dengan meningkatkan fungsi Anda hingga batas konkurensi maksimum yang dikonfigurasi. Setelah batas itu tercapai, Lambda merespons permintaan tambahan dengan respons HTTP 429. Tidak ada mekanisme antrian bawaan, jadi penanganan penskalaan sepenuhnya bergantung pada konfigurasi Lambda Event Function Anda. Secara default, Lambda Event Functions memiliki batas 1.000 eksekusi bersamaan per. Wilayah AWS
- API Gateway menyediakan kemampuan penskalaan tambahan di atas penskalaan Lambda sendiri. Ini termasuk kontrol antrian permintaan dan pelambatan bawaan, memungkinkan Anda mengelola lonjakan lalu lintas dengan lebih anggun. API Gateway dapat menangani hingga 10.000 permintaan per detik per wilayah secara default, dengan kapasitas burst 5.000 permintaan per detik. Ini juga menyediakan alat untuk membatasi permintaan pada tingkat yang berbeda (API, panggung, atau metode) untuk melindungi backend Anda.

Memantau

- URL fungsi menawarkan pemantauan dasar melalui CloudWatch metrik Amazon, termasuk jumlah permintaan, latensi, dan tingkat kesalahan. Anda mendapatkan akses ke metrik dan log Lambda standar, yang menunjukkan permintaan mentah yang masuk ke fungsi Anda. Meskipun ini memberikan visibilitas operasional yang penting, metrik difokuskan terutama pada eksekusi fungsi.
- API Gateway menyediakan kemampuan pemantauan komprehensif termasuk metrik terperinci, pencatatan, dan opsi penelusuran. Anda dapat memantau panggilan API, latensi, tingkat kesalahan, dan hit/miss tingkat cache melalui CloudWatch. API Gateway juga terintegrasi dengan AWS X-Ray untuk penelusuran terdistribusi dan menyediakan format logging yang dapat disesuaikan.

Biaya

- URL fungsi mengikuti model penetapan harga Lambda standar — Anda hanya membayar untuk pemanggilan fungsi dan waktu komputasi. Tidak ada biaya tambahan untuk titik akhir URL itu sendiri. Ini menjadikannya pilihan hemat biaya untuk API sederhana atau aplikasi dengan lalu lintas rendah jika Anda tidak memerlukan fitur tambahan API Gateway.
- API Gateway menawarkan tingkat gratis yang mencakup satu juta panggilan API yang diterima untuk REST API dan satu juta panggilan API yang diterima untuk API HTTP. Setelah ini, API Gateway mengenakan biaya untuk panggilan API, transfer data, dan caching (jika diaktifkan). Lihat halaman harga API Gateway untuk memahami biaya kasus penggunaan Anda sendiri.

Fitur lainnya

- URL fungsi dirancang untuk kesederhanaan dan integrasi Lambda langsung. Mereka mendukung titik akhir HTTP dan HTTPS, menawarkan dukungan CORS bawaan, dan menyediakan titik akhir dual-stack (IPv4 dan IPv6). Meskipun mereka tidak memiliki fitur canggih, mereka unggul dalam skenario di mana Anda memerlukan cara cepat dan mudah untuk mengekspos Fungsi Peristiwa Lambda melalui HTTP.
- API Gateway mencakup banyak fitur tambahan seperti versi API, manajemen tahap, kunci API untuk paket penggunaan, dokumentasi API melalui, API Swagger/OpenAPI, WebSocket API pribadi dalam VPC, dan integrasi WAF untuk keamanan tambahan. Ini juga mendukung penerapan kenari, integrasi tiruan untuk pengujian, dan integrasi dengan yang lain di luar Lambda. Layanan AWS

Pilih metode untuk menjalankan Fungsi Peristiwa Lambda Anda

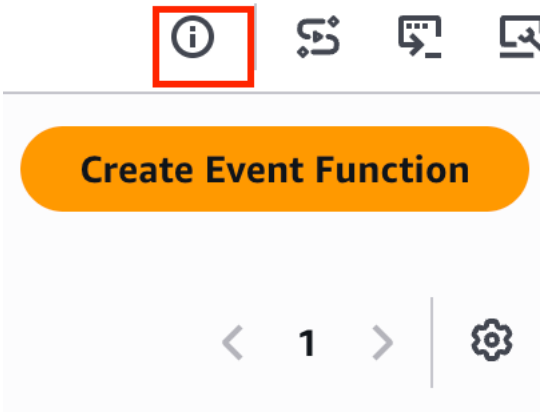
Sekarang setelah Anda membaca tentang kriteria untuk memilih antara URL fungsi Lambda dan API Gateway dan perbedaan utama di antara keduanya, Anda dapat memilih opsi yang paling sesuai dengan kebutuhan Anda dan menggunakan sumber daya berikut untuk membantu Anda mulai menggunakannya.

Function URLs

Memulai URL fungsi dengan sumber daya berikut

- Ikuti tutorial Membuat Fungsi Acara Lambda dengan URL fungsi
- Pelajari lebih lanjut tentang URL fungsi di the section called “URL fungsi” bagian panduan ini
- Coba tutorial yang dipandu dalam konsol Buat aplikasi web sederhana dengan melakukan hal berikut:

1. Buka halaman Fungsi Acara konsol Lambda.
2. Buka panel bantuan dengan memilih ikon di sudut kanan atas layar.



3. Pilih Tutorial.
4. Di Buat aplikasi web sederhana, pilih Mulai tutorial.

API Gateway

Memulai Lambda dan API Gateway dengan sumber daya berikut

- Ikuti tutorial Menggunakan Lambda dengan API Gateway untuk membuat REST API yang terintegrasi dengan Fungsi Acara Lambda backend.

- Pelajari selengkapnya tentang berbagai jenis API yang ditawarkan oleh API Gateway di bagian berikut dari Panduan Pengembang Amazon API Gateway:
  - API API Gateway REST
  - API API Gateway HTTP
  - API Gateway WebSocket API
- Coba satu atau beberapa contoh di bagian Tutorial dan lokakarya dari Panduan Pengembang Amazon API Gateway.

Penggunaan AWS Lambda dengan AWS Infrastructure Composer

AWS Infrastructure Composer adalah pembangun visual untuk merancang aplikasi modern. AWS Anda mendesain arsitektur aplikasi Anda dengan menyeret, mengelompokkan, dan menghubungkan Layanan AWS dalam kanvas visual. Infrastructure Composer membuat template infrastruktur sebagai kode (IAc) dari desain Anda yang dapat Anda terapkan menggunakan atau. AWS SAMCloudFormation

Mengekspor Fungsi Acara Lambda ke Komposer Infrastruktur

Anda dapat mulai menggunakan Infrastructure Composer dengan membuat proyek baru berdasarkan konfigurasi Fungsi Peristiwa Lambda yang ada menggunakan konsol Lambda. Untuk mengekspor konfigurasi dan kode fungsi Anda ke Infrastructure Composer untuk membuat proyek baru, lakukan hal berikut:

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih fungsi yang ingin Anda gunakan sebagai dasar untuk proyek Infrastructure Composer Anda.
3. Di panel Ikhtisar fungsi, pilih Ekspor ke Komposer Infrastruktur.

   Untuk mengekspor konfigurasi dan kode fungsi Anda ke Infrastructure Composer, Lambda membuat bucket Amazon S3 di akun Anda untuk menyimpan sementara data ini.
4. Di kotak dialog, pilih Konfirmasi dan buat proyek untuk menerima nama default untuk bucket ini dan ekspor konfigurasi dan kode fungsi Anda ke Infrastructure Composer.
5. (Opsional) Untuk memilih nama lain untuk bucket Amazon S3 yang dibuat Lambda, masukkan nama baru dan pilih Konfirmasi dan buat proyek. Nama bucket Amazon S3 harus unik secara global dan mengikuti aturan penamaan bucket.
6. Untuk menyimpan file proyek dan fungsi Anda di Infrastructure Composer, aktifkan mode sinkronisasi lokal.

> **Note**
>
> Jika Anda pernah menggunakan fitur Ekspor ke Komposer Aplikasi sebelumnya dan membuat bucket Amazon S3 menggunakan nama default, Lambda dapat menggunakan kembali bucket ini jika masih ada. Terima nama bucket default di kotak dialog untuk menggunakan kembali bucket yang ada.

Konfigurasi bucket transfer Amazon S3

Bucket Amazon S3 yang dibuat Lambda untuk mentransfer konfigurasi fungsi Anda secara otomatis mengenkripsi objek menggunakan standar enkripsi AES 256. Lambda juga mengonfigurasi bucket untuk menggunakan kondisi pemilik bucket untuk memastikan bahwa hanya Anda Akun AWS yang dapat menambahkan objek ke bucket.

Lambda mengonfigurasi bucket untuk menghapus objek secara otomatis 10 hari setelah diunggah. Namun, Lambda tidak secara otomatis menghapus bucket itu sendiri. Untuk menghapus ember dari Anda Akun AWS, ikuti petunjuk di Menghapus ember. Nama bucket default menggunakan awalan`lambdasam`, string alfanumerik 10 digit, dan fungsi yang Wilayah AWS Anda buat di:

```
lambdasam-06f22da95b-us-east-1
```

Untuk menghindari biaya tambahan ditambahkan ke Anda Akun AWS, kami sarankan Anda menghapus bucket Amazon S3 segera setelah Anda selesai mengekspor fungsi Anda ke Infrastructure Composer.

Harga Amazon S3 standar berlaku.

Izin yang diperlukan

Untuk menggunakan integrasi Lambda dengan fitur Infrastructure Composer, Anda memerlukan izin tertentu untuk mengunduh AWS SAM templat dan menulis konfigurasi fungsi Anda ke Amazon S3.

Untuk mengunduh AWS SAM templat, Anda harus memiliki izin untuk menggunakan tindakan API berikut:

- GetPolicy
- saya: GetPolicyVersion
- saya: GetRole

- saya: GetRolePolicy
- saya: ListAttachedRolePolicies
- saya: ListRolePolicies
- saya: ListRoles

Anda dapat memberikan izin untuk menggunakan semua tindakan ini dengan menambahkan kebijakan Lambda_ReadOnlyAccess AWS terkelola AWS ke peran pengguna IAM Anda.

Agar Lambda dapat menulis konfigurasi fungsi Anda ke Amazon S3, Anda harus memiliki izin untuk menggunakan tindakan API berikut:

- S3:PutObject
- S3:CreateBucket
- S3:PutBucketEncryption
- S3:PutBucketLifecycleConfiguration

Jika Anda tidak dapat mengekspor konfigurasi fungsi Anda ke Infrastructure Composer, periksa apakah akun Anda memiliki izin yang diperlukan untuk operasi ini. Jika Anda memiliki izin yang diperlukan, tetapi masih tidak dapat mengekspor konfigurasi fungsi Anda, periksa kebijakan berbasis sumber daya yang mungkin membatasi akses ke Amazon S3.

Sumber daya lainnya

Untuk tutorial yang lebih rinci tentang cara merancang aplikasi tanpa server di Infrastructure Composer berdasarkan Fungsi Peristiwa Lambda yang ada, lihat. the section called “Infrastruktur sebagai kode (IAc)”

Untuk menggunakan Infrastructure Composer dan AWS SAM untuk merancang dan menyebarkan aplikasi tanpa server lengkap menggunakan Lambda, Anda juga dapat mengikuti AWS Infrastructure Composer tutorial di Workshop Pola Tanpa Server.AWS

Penggunaan AWS Lambda dengan CloudFormation

Dalam AWS CloudFormation template, Anda dapat menentukan Fungsi Peristiwa Lambda sebagai target sumber daya kustom. Gunakan sumber daya khusus untuk memproses parameter, mengambil nilai konfigurasi, atau memanggil lainnya Layanan AWS selama peristiwa siklus hidup tumpukan.

Contoh berikut memanggil fungsi yang ditentukan di bagian lain dalam templat.

Example– Penentuan sumber daya kustom

```
Resources:
  primerinvoke:
    Type: AWS::CloudFormation::CustomResource
    Version: "1.0"
    Properties:
      ServiceToken: !GetAtt primer.Arn
      FunctionName: !Ref randomerror
```

Token layanan adalah Nama Sumber Daya Amazon (ARN) dari fungsi yang CloudFormation dipanggil saat Anda membuat, memperbarui, atau menghapus tumpukan. Anda juga dapat menyertakan properti tambahan seperti`FunctionName`, yang CloudFormation lolos ke fungsi Anda apa adanya.

CloudFormation memanggil Fungsi Peristiwa Lambda Anda secara asinkron dengan peristiwa yang menyertakan URL panggilan balik.

Example– CloudFormation pesan acara

```
{
    "RequestType": "Create",
    "ServiceToken": "arn:aws:lambda:us-east-1:123456789012:function:lambda-error-
processor-primer-14ROR2T3JKU66",
    "ResponseURL": "https://cloudformation-custom-resource-response-useast1.s3-us-
east-1.amazonaws.com/arn%3Aaws%3Acloudformation%3Aus-east-1%3A123456789012%3Astack/
lambda-error-processor/1134083a-2608-1e91-9897-022501a2c456%7Cprimerinvoke
%7C5d478078-13e9-baf0-464a-7ef285ecc786?
AWSAccessKeyId=AKIAIOSFODNN7EXAMPLE&Expires=1555451971&Signature=28UijZePE5I4dvukKQqM
%2F9Rf1o4%3D",
    "StackId": "arn:aws:cloudformation:us-east-1:123456789012:stack/lambda-error-
processor/1134083a-2608-1e91-9897-022501a2c456",
    "RequestId": "5d478078-13e9-baf0-464a-7ef285ecc786",
    "LogicalResourceId": "primerinvoke",
```

```
    "ResourceType": "AWS::CloudFormation::CustomResource",
    "ResourceProperties": {
        "ServiceToken": "arn:aws:lambda:us-east-1:123456789012:function:lambda-error-
processor-primer-14ROR2T3JKU66",
        "FunctionName": "lambda-error-processor-randomerror-ZWUC391MQAJK"
    }
}
```

Fungsi ini bertanggung jawab untuk mengembalikan respons ke URL callback yang menunjukkan keberhasilan atau kegagalan. Untuk sintaksis respons lengkap, lihat Objek respons sumber daya kustom.

Example– CloudFormation respon sumber daya kustom

```
{
    "Status": "SUCCESS",
    "PhysicalResourceId": "2019/04/18/[$LATEST]b3d1bfc65f19ec610654e4d9b9de47a0",
    "StackId": "arn:aws:cloudformation:us-east-1:123456789012:stack/lambda-error-
processor/1134083a-2608-1e91-9897-022501a2c456",
    "RequestId": "5d478078-13e9-baf0-464a-7ef285ecc786",
    "LogicalResourceId": "primerinvoke"
}
```

CloudFormation menyediakan perpustakaan bernama `cfn-response` yang menangani pengiriman respons. Jika Anda menentukan fungsi Anda dalam template, Anda dapat meminta perpustakaan berdasarkan nama. CloudFormation kemudian menambahkan pustaka ke paket penerapan yang dibuatnya untuk fungsi tersebut.

Jika fungsi yang digunakan Sumber Daya Kustom memiliki Antarmuka Jaringan Elastis yang melekat padanya, tambahkan sumber daya berikut ke kebijakan VPC di mana **`region`** Wilayah fungsi berada tanpa tanda hubung. Sebagai contoh, `us-east-1` adalah`useast1`. Ini akan memungkinkan Custom Resource untuk merespons URL callback yang mengirimkan sinyal kembali ke CloudFormation tumpukan.

```
arn:aws:s3:::cloudformation-custom-resource-response-region",
"arn:aws:s3:::cloudformation-custom-resource-response-region/*",
```

Contoh fungsi berikut memanggil suatu fungsi kedua. Jika panggilan berhasil, fungsi mengirimkan respons sukses CloudFormation, dan pembaruan tumpukan berlanjut. Template menggunakan tipe sumber daya AWS::Serverless: :Function yang disediakan oleh AWS Serverless Application Model.

Example— Fungsi sumber daya khusus

```
Transform: 'AWS::Serverless-2016-10-31'
Resources:
  primer:
    Type: AWS::Serverless::Function
    Properties:
      Handler: index.handler
      Runtime: nodejs16.x
      InlineCode: |
        var aws = require('aws-sdk');
        var response = require('cfn-response');
        exports.handler = function(event, context) {
            // For Delete requests, immediately send a SUCCESS response.
            if (event.RequestType == "Delete") {
                response.send(event, context, "SUCCESS");
                return;
            }
            var responseStatus = "FAILED";
            var responseData = {};
            var functionName = event.ResourceProperties.FunctionName
            var lambda = new aws.Lambda();
            lambda.invoke({ FunctionName: functionName }, function(err, invokeResult) {
                if (err) {
                    responseData = {Error: "Invoke call failed"};
                    console.log(responseData.Error + ":\n", err);
                }
                else responseStatus = "SUCCESS";
                response.send(event, context, responseStatus, responseData);
            });
        };
      Description: Invoke a function to create a log stream.
      MemorySize: 128
      Timeout: 8
      Role: !GetAtt role.Arn
      Tracing: Active
```

Jika fungsi yang dipanggil sumber daya kustom tidak ditentukan dalam templat, Anda bisa mendapatkan kode sumbernya `cfn-response` dari modul cfn-response di Panduan Pengguna. AWS CloudFormation

Untuk informasi selengkapnya tentang sumber daya kustom, lihat Sumber daya kustom dalam Panduan Pengguna AWS CloudFormation .

Memproses acara Amazon DocumentDB dengan Lambda

Anda dapat menggunakan Fungsi Peristiwa Lambda untuk memproses peristiwa di aliran perubahan Amazon DocumentDB (dengan kompatibilitas MongoDB) dengan mengonfigurasi cluster Amazon DocumentDB sebagai sumber peristiwa. Kemudian, Anda dapat mengotomatiskan beban kerja berbasis peristiwa dengan menjalankan Fungsi Peristiwa Lambda setiap kali data berubah dengan cluster Amazon DocumentDB Anda.

> **Note**
>
> Lambda hanya mendukung Amazon DocumentDB versi 4.0 dan 5.0. Lambda tidak mendukung versi 3.6.
> Selain itu, untuk pemetaan sumber acara, Lambda mendukung cluster berbasis instance dan cluster regional saja. Lambda tidak mendukung cluster elastis atau cluster global. Batasan ini tidak berlaku saat menggunakan Lambda sebagai klien untuk terhubung ke Amazon DocumentDB. Lambda dapat terhubung ke semua jenis cluster untuk melakukan operasi CRUD.

Lambda memproses peristiwa dari Amazon DocumentDB mengubah aliran secara berurutan sesuai urutan kedatangannya. Karena itu, fungsi Anda hanya dapat menangani satu pemanggilan bersamaan dari Amazon DocumentDB sekaligus. Untuk memantau fungsi Anda, Anda dapat melacak metrik konkurensinya.

> **Warning**
>
> Pemetaan sumber peristiwa Lambda memproses setiap peristiwa setidaknya sekali, dan pemrosesan duplikat catatan dapat terjadi. Untuk menghindari potensi masalah yang terkait dengan duplikat peristiwa, kami sangat menyarankan agar Anda membuat kode fungsi Anda idempoten. Untuk mempelajari lebih lanjut, lihat Bagaimana cara membuat fungsi Lambda saya idempoten di Pusat Pengetahuan. AWS

Topik

- Contoh acara Amazon DocumentDB
- Prasyarat dan izin
- Konfigurasikan keamanan jaringan

- Membuat pemetaan sumber peristiwa Amazon DocumentDB (konsol)
- Membuat pemetaan sumber peristiwa Amazon DocumentDB (SDK atau CLI)
- Posisi awal polling dan streaming
- Memantau sumber acara Amazon DocumentDB Anda
- Tutorial: Menggunakan AWS Lambda dengan Amazon DocumentDB Streams

Contoh acara Amazon DocumentDB

```
{
    "eventSourceArn": "arn:aws:rds:us-
east-1:123456789012:cluster:canaryclusterb2a659a2-qo5tcmqkcl03",
    "events": [
        {
            "event": {
                "_id": {
                    "_data": "0163eeb6e7000000090100000009000041e1"
                },
                "clusterTime": {
                    "$timestamp": {
                        "t": 1676588775,
                        "i": 9
                    }
                },
                "documentKey": {
                    "_id": {
                        "$oid": "63eeb6e7d418cd98afb1c1d7"
                    }
                },
                "fullDocument": {
                    "_id": {
                        "$oid": "63eeb6e7d418cd98afb1c1d7"
                    },
                    "anyField": "sampleValue"
                },
                "ns": {
                    "db": "test_database",
                    "coll": "test_collection"
                },
                "operationType": "insert"
            }
        }
```

```
    ],
    "eventSource": "aws:docdb"
}
```

Untuk informasi selengkapnya tentang peristiwa dalam contoh ini dan bentuknya, lihat Mengubah Acara di situs web Dokumentasi MongoDB.

Prasyarat dan izin

Sebelum Anda dapat menggunakan Amazon DocumentDB sebagai sumber acara untuk Fungsi Acara Lambda Anda, perhatikan prasyarat berikut. Anda harus:

- Memiliki cluster Amazon DocumentDB yang ada di tempat yang Akun AWS sama Wilayah AWS dan sebagai fungsi Anda. Jika Anda tidak memiliki cluster yang ada, Anda dapat membuatnya dengan mengikuti langkah-langkah di Memulai dengan Amazon DocumentDB di Panduan Pengembang Amazon DocumentDB. Atau, serangkaian langkah pertama dalam Tutorial: Menggunakan AWS Lambda dengan Amazon DocumentDB Streams memandu Anda membuat cluster Amazon DocumentDB dengan semua prasyarat yang diperlukan.
- Izinkan Lambda mengakses sumber daya Amazon Virtual Private Cloud (Amazon VPC) yang terkait dengan cluster Amazon DocumentDB Anda. Untuk informasi selengkapnya, lihat Konfigurasikan keamanan jaringan.
- Aktifkan TLS di klaster Amazon DocumentDB Anda. Ini adalah pengaturan default. Jika Anda menonaktifkan TLS, maka Lambda tidak dapat berkomunikasi dengan cluster Anda.
- Aktifkan aliran perubahan di cluster Amazon DocumentDB Anda. Untuk informasi selengkapnya, lihat Menggunakan Ubah Aliran dengan Amazon DocumentDB di Panduan Pengembang Amazon DocumentDB.
- Berikan Lambda kredenal untuk mengakses klaster Amazon DocumentDB Anda. Saat menyiapkan sumber acara, berikan AWS Secrets Managerkunci yang berisi detail otentikasi (nama pengguna dan kata sandi) yang diperlukan untuk mengakses klaster Anda. Untuk menyediakan kunci ini selama penyiapan, lakukan salah satu hal berikut:
  - Jika Anda menggunakan konsol Lambda untuk penyiapan, berikan kunci di bidang kunci manajer Rahasia.
  - Jika Anda menggunakan AWS Command Line Interface (AWS CLI) untuk penyiapan, berikan kunci ini di `source-access-configurations` opsi. Anda dapat menyertakan opsi ini dengan `create-event-source-mapping`perintah atau `update-event-source-mapping`perintah. Contoh:

```
aws lambda create-event-source-mapping \
    ...
    --source-access-configurations
 '[{"Type":"BASIC_AUTH","URI":"arn:aws:secretsmanager:us-
west-2:123456789012:secret:DocDBSecret-AbC4E6"}]' \
    ...
```

- Berikan izin Lambda untuk mengelola sumber daya yang terkait dengan aliran Amazon DocumentDB Anda. Tambahkan izin berikut secara manual ke peran eksekusi fungsi Anda:
  - RDS: DescribedBClusters
  - rds: DescribeDBClusterParameters
  - rds: DescribeDBSubnetGroups
  - EC2: CreateNetworkInterface
  - EC2: DescribeNetworkInterfaces
  - EC2: DescribeVpcs
  - EC2: DeleteNetworkInterface
  - EC2: DescribeSubnets
  - EC2: DescribeSecurityGroups
  - kms:Decrypt
  - manajer rahasia: GetSecretValue
- Pertahankan ukuran peristiwa aliran perubahan Amazon DocumentDB yang Anda kirim ke Lambda di bawah 6 MB. Lambda mendukung ukuran payload hingga 6 MB. Jika aliran perubahan Anda mencoba mengirim Lambda peristiwa yang lebih besar dari 6 MB, maka Lambda akan menghapus pesan dan memancarkan metrik. `OversizedRecordCount` Lambda memancarkan semua metrik dengan upaya terbaik.

> **Note**
>
> Sementara Fungsi Acara Lambda biasanya memiliki batas waktu tunggu maksimum 15 menit, pemetaan sumber acara untuk Amazon MSK, Apache Kafka yang dikelola sendiri, Amazon DocumentDB, dan Amazon MQ untuk ActiveMQ dan RabbitMQ hanya mendukung fungsi dengan batas waktu tunggu maksimum 14 menit. Kendala ini memastikan bahwa

pemetaan sumber peristiwa dapat menangani kesalahan fungsi dan percobaan ulang dengan benar.

Konfigurasikan keamanan jaringan

Untuk memberi Lambda akses penuh ke Amazon DocumentDB melalui pemetaan sumber peristiwa, kluster Anda harus menggunakan titik akhir publik (alamat IP publik), atau Anda harus memberikan akses ke Amazon VPC tempat Anda membuat klaster.

Saat Anda menggunakan Amazon DocumentDB dengan Lambda, AWS PrivateLink buat titik akhir VPC yang menyediakan akses Fungsi Acara ke sumber daya di Amazon VPC Anda.

> **Note**
>
> AWS PrivateLink Titik akhir VPC diperlukan untuk fungsi dengan pemetaan sumber peristiwa yang menggunakan mode default (sesuai permintaan) untuk poller acara. Jika pemetaan sumber acara menggunakan mode yang disediakan, Anda tidak perlu mengonfigurasi titik akhir AWS PrivateLink VPC.

Buat titik akhir untuk menyediakan akses ke sumber daya berikut:

- Lambda — Buat titik akhir untuk kepala layanan Lambda.
- AWS STS — Buat titik akhir AWS STS untuk prinsipal layanan untuk mengambil peran atas nama Anda.
- Secrets Manager - Jika klaster Anda menggunakan Secrets Manager untuk menyimpan kredensi, buat endpoint untuk Secrets Manager.

Atau, konfigurasikan gateway NAT di setiap subnet publik di Amazon VPC. Untuk informasi selengkapnya, lihat the section called “Akses internet untuk fungsi VPC”.

Saat Anda membuat pemetaan sumber peristiwa untuk Amazon DocumentDB, Lambda memeriksa apakah Antarmuka Jaringan Elastis (ENI) sudah ada untuk subnet dan grup keamanan yang dikonfigurasi untuk Amazon VPC Anda. Jika Lambda menemukan ENI yang ada, ia mencoba untuk menggunakannya kembali. Jika tidak, Lambda membuat ENI baru untuk terhubung ke sumber acara dan menjalankan fungsi Anda.

> **Note**
>
> Fungsi Acara Lambda selalu berjalan di dalam VPC yang dimiliki oleh layanan Lambda. Konfigurasi VPC fungsi Anda tidak memengaruhi pemetaan sumber peristiwa. Hanya konfigurasi jaringan sumber acara yang menentukan bagaimana Lambda terhubung ke sumber acara Anda.

Konfigurasikan grup keamanan untuk VPC Amazon yang berisi cluster Anda. Secara default, Amazon DocumentDB menggunakan port berikut:. `27017`

- Aturan masuk - Izinkan semua lalu lintas di port broker default untuk grup keamanan yang terkait dengan sumber acara Anda. Atau, Anda dapat menggunakan aturan grup keamanan referensi mandiri untuk mengizinkan akses dari instans dalam grup keamanan yang sama.
- Aturan keluar — Izinkan semua lalu lintas di port `443` untuk tujuan eksternal jika fungsi Anda perlu berkomunikasi dengan AWS layanan. Atau, Anda juga dapat menggunakan aturan grup keamanan referensi diri untuk membatasi akses ke broker jika Anda tidak perlu berkomunikasi dengan layanan lain AWS .
- Aturan masuk titik akhir VPC Amazon — Jika Anda menggunakan titik akhir VPC Amazon, grup keamanan yang terkait dengan titik akhir VPC Amazon Anda harus mengizinkan lalu lintas masuk pada port dari grup keamanan klaster. `443`

Jika klaster menggunakan autentikasi, Anda juga dapat membatasi kebijakan endpoint untuk titik akhir Secrets Manager. Untuk memanggil Secrets Manager API, Lambda menggunakan peran fungsi Anda, bukan kepala layanan Lambda.

Example Kebijakan titik akhir VPC — titik akhir Secrets Manager

```
{
      "Statement": [
          {
              "Action": "secretsmanager:GetSecretValue",
              "Effect": "Allow",
              "Principal": {
                  "AWS": [
                      "arn:aws::iam::123456789012:role/my-role"
                  ]
              },
```

```
            "Resource": "arn:aws::secretsmanager:us-west-2:123456789012:secret:my-
secret"
        }
    ]
}
```

Saat Anda menggunakan titik akhir Amazon VPC, AWS merutekan panggilan API Anda untuk menjalankan fungsi Anda menggunakan Antarmuka Jaringan Elastis (ENI) titik akhir. Kepala layanan Lambda perlu memanggil `lambda:InvokeFunction` peran dan fungsi apa pun yang menggunakan ENI tersebut.

Secara default, titik akhir Amazon VPC memiliki kebijakan IAM terbuka yang memungkinkan akses luas ke sumber daya. Praktik terbaik adalah membatasi kebijakan ini untuk melakukan tindakan yang diperlukan menggunakan titik akhir tersebut. Untuk memastikan bahwa pemetaan sumber peristiwa Anda dapat menjalankan Fungsi Peristiwa Lambda Anda, kebijakan titik akhir VPC harus mengizinkan kepala layanan Lambda untuk memanggil dan. `sts:AssumeRole` `lambda:InvokeFunction` Membatasi kebijakan titik akhir VPC Anda agar hanya mengizinkan panggilan API yang berasal dari organisasi Anda mencegah pemetaan sumber peristiwa berfungsi dengan baik, `"Resource": "*"` sehingga diperlukan dalam kebijakan ini.

Contoh kebijakan titik akhir VPC berikut menunjukkan cara memberikan akses yang diperlukan ke prinsipal layanan Lambda untuk titik akhir dan Lambda. AWS STS

Example Kebijakan Titik Akhir VPC — AWS STS titik akhir

```
{
    "Statement": [
        {
            "Action": "sts:AssumeRole",
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "lambda.amazonaws.com"
                ]
            },
            "Resource": "*"
        }
    ]
}
```

Example Kebijakan Titik Akhir VPC - Titik akhir Lambda

```
{
      "Statement": [
          {
              "Action": "lambda:InvokeFunction",
              "Effect": "Allow",
              "Principal": {
                  "Service": [
                      "lambda.amazonaws.com"
                  ]
              },
              "Resource": "*"
          }
      ]
  }
```

Membuat pemetaan sumber peristiwa Amazon DocumentDB (konsol)

Agar Fungsi Peristiwa Lambda dapat dibaca dari aliran perubahan klaster Amazon DocumentDB, buat pemetaan sumber peristiwa. Bagian ini menjelaskan cara melakukan ini dari konsol Lambda. Untuk AWS SDK dan AWS CLI instruksi, lihatthe section called “Membuat pemetaan sumber peristiwa Amazon DocumentDB (SDK atau CLI)”.

Untuk membuat pemetaan sumber peristiwa Amazon DocumentDB (konsol)

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih nama sebuah fungsi.
3. Di bagian Gambaran umum fungsi, pilih Tambah pemicu.
4. Di bawah konfigurasi Trigger, dalam daftar dropdown, pilih DocumentDB.
5. Konfigurasikan opsi yang diperlukan, lalu pilih Tambah.

Lambda mendukung opsi berikut untuk sumber acara Amazon DocumentDB:

- Cluster DocumentDB - Pilih cluster Amazon DocumentDB.
- Aktifkan pemicu - Pilih apakah Anda ingin segera mengaktifkan pemicu. Jika Anda memilih kotak centang ini, maka fungsi Anda segera mulai menerima lalu lintas dari aliran perubahan Amazon DocumentDB yang ditentukan setelah pembuatan pemetaan sumber peristiwa. Kami menyarankan Anda mengosongkan kotak centang untuk membuat pemetaan sumber peristiwa dalam status

dinonaktifkan untuk pengujian. Setelah pembuatan, Anda dapat mengaktifkan pemetaan sumber acara kapan saja.

- Nama database — Masukkan nama database dalam cluster untuk dikonsumsi.
- (Opsional) Nama koleksi — Masukkan nama koleksi dalam database untuk dikonsumsi. Jika Anda tidak menentukan koleksi, maka Lambda akan mendengarkan semua peristiwa dari setiap koleksi dalam database.
- Ukuran Batch - Atur jumlah maksimum pesan yang akan diambil dalam satu batch, hingga 10.000. Ukuran batch default adalah 100.
- Posisi awal — Pilih posisi dalam aliran untuk mulai membaca catatan dari.
  - Terbaru - Memproses hanya catatan baru yang ditambahkan ke aliran. Fungsi Anda mulai memproses catatan hanya setelah Lambda selesai membuat sumber acara Anda. Ini berarti bahwa beberapa catatan mungkin dijatuhkan sampai sumber acara Anda berhasil dibuat.
  - Trim horizon – Memproses semua rekaman dalam aliran. Lambda menggunakan durasi retensi log klaster Anda untuk menentukan dari mana harus mulai membaca peristiwa. Secara khusus, Lambda mulai membaca dari. `current_time - log_retention_duration` Aliran perubahan Anda harus sudah aktif sebelum stempel waktu ini agar Lambda dapat membaca semua peristiwa dengan benar.
  - Pada stempel waktu – Proses rekaman dimulai dari waktu tertentu. Aliran perubahan Anda harus sudah aktif sebelum stempel waktu yang ditentukan agar Lambda dapat membaca semua peristiwa dengan benar.
- Otentikasi — Pilih metode otentikasi untuk mengakses broker di cluster Anda.
  - BASIC_AUTH — Dengan otentikasi dasar, Anda harus memberikan kunci Secrets Manager yang berisi kredensi untuk mengakses klaster Anda.
- Kunci Secrets Manager - Pilih kunci Secrets Manager yang berisi detail otentikasi (nama pengguna dan kata sandi) yang diperlukan untuk mengakses klaster Amazon DocumentDB Anda.
- (Opsional) Jendela Batch - Atur jumlah waktu maksimum dalam hitungan detik untuk mengumpulkan catatan sebelum menjalankan fungsi Anda, hingga 300.
- (Opsional) Konfigurasi dokumen lengkap — Untuk operasi pembaruan dokumen, pilih apa yang ingin Anda kirim ke aliran. Nilai defaultnya adalah`Default`, yang berarti bahwa untuk setiap peristiwa aliran perubahan, Amazon DocumentDB hanya mengirimkan delta yang menjelaskan perubahan yang dibuat. Untuk informasi selengkapnya tentang bidang ini, lihat FullDocumentdi dokumentasi MongoDB Javadoc API.
  - Default — Lambda hanya mengirimkan sebagian dokumen yang menjelaskan perubahan yang dibuat.

- UpdateLookup— Lambda mengirimkan delta yang menjelaskan perubahan, bersama dengan salinan seluruh dokumen.

Membuat pemetaan sumber peristiwa Amazon DocumentDB (SDK atau CLI)

Untuk membuat atau mengelola pemetaan sumber peristiwa Amazon DocumentDB dengan SDK, Anda dapat AWS menggunakan operasi API berikut:

- CreateEventSourceMapping
- ListEventSourceMappings
- GetEventSourceMapping
- UpdateEventSourceMapping
- DeleteEventSourceMapping

Untuk membuat pemetaan sumber acara dengan AWS CLI, gunakan `create-event-source-mapping`perintah. Contoh berikut menggunakan perintah ini untuk memetakan fungsi bernama `my-function` ke aliran perubahan Amazon DocumentDB. Sumber peristiwa ditentukan oleh Amazon Resource Name (ARN), dengan ukuran batch 500, mulai dari stempel waktu dalam waktu Unix. Perintah ini juga menentukan kunci Secrets Manager yang digunakan Lambda untuk terhubung ke Amazon DocumentDB. Selain itu, ini termasuk `document-db-event-source-config` parameter yang menentukan database dan koleksi untuk dibaca.

```
aws lambda create-event-source-mapping --function-name my-function \
    --event-source-arn arn:aws:rds:us-west-2:123456789012:cluster:privatecluster7de2-
epzcyvu4pjoy
    --batch-size 500 \
    --starting-position AT_TIMESTAMP \
    --starting-position-timestamp 1541139109 \
    --source-access-configurations
 '[{"Type":"BASIC_AUTH","URI":"arn:aws:secretsmanager:us-
east-1:123456789012:secret:DocDBSecret-BAtjxi"}]' \
    --document-db-event-source-config '{"DatabaseName":"test_database",
 "CollectionName": "test_collection"}' \
```

Anda akan melihat output seperti ini:

```
{
    "UUID": "2b733gdc-8ac3-cdf5-af3a-1827b3b11284",
```

```
    "BatchSize": 500,
    "DocumentDBEventSourceConfig": {
        "CollectionName": "test_collection",
        "DatabaseName": "test_database",
        "FullDocument": "Default"
    },
    "MaximumBatchingWindowInSeconds": 0,
    "EventSourceArn": "arn:aws:rds:us-west-2:123456789012:cluster:privatecluster7de2-
epzcyvu4pjoy",
    "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-function",
    "LastModified": 1541348195.412,
    "LastProcessingResult": "No records processed",
    "State": "Creating",
    "StateTransitionReason": "User action"
}
```

Setelah pembuatan, Anda dapat menggunakan `update-event-source-mapping`perintah untuk memperbarui pengaturan untuk sumber acara Amazon DocumentDB Anda. Contoh berikut memperbarui ukuran batch menjadi 1.000 dan jendela batch menjadi 10 detik. Untuk perintah ini, Anda memerlukan UUID pemetaan sumber acara Anda, yang dapat Anda ambil menggunakan perintah `list-event-source-mapping` atau konsol Lambda.

```
aws lambda update-event-source-mapping --function-name my-function \
    --uuid f89f8514-cdd9-4602-9e1f-01a5b77d449b \
    --batch-size 1000 \
    --batch-window 10
```

Anda akan melihat output ini yang terlihat seperti ini:

```
{
    "UUID": "2b733gdc-8ac3-cdf5-af3a-1827b3b11284",
    "BatchSize": 500,
    "DocumentDBEventSourceConfig": {
        "CollectionName": "test_collection",
        "DatabaseName": "test_database",
        "FullDocument": "Default"
    },
    "MaximumBatchingWindowInSeconds": 0,
    "EventSourceArn": "arn:aws:rds:us-west-2:123456789012:cluster:privatecluster7de2-
epzcyvu4pjoy",
    "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-function",
    "LastModified": 1541359182.919,
```

```
    "LastProcessingResult": "OK",
    "State": "Updating",
    "StateTransitionReason": "User action"
}
```

Lambda memperbarui pengaturan secara asinkron, jadi Anda mungkin tidak melihat perubahan ini dalam output sampai proses selesai. Untuk melihat pengaturan saat ini dari pemetaan sumber acara Anda, gunakan `get-event-source-mapping`perintah.

```
aws lambda get-event-source-mapping --uuid f89f8514-cdd9-4602-9e1f-01a5b77d449b
```

Anda akan melihat output ini yang terlihat seperti ini:

```
{
    "UUID": "2b733gdc-8ac3-cdf5-af3a-1827b3b11284",
    "DocumentDBEventSourceConfig": {
        "CollectionName": "test_collection",
        "DatabaseName": "test_database",
        "FullDocument": "Default"
    },
    "BatchSize": 1000,
    "MaximumBatchingWindowInSeconds": 10,
    "EventSourceArn": "arn:aws:rds:us-west-2:123456789012:cluster:privatecluster7de2-
epzcyvu4pjoy",
    "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-function",
    "LastModified": 1541359182.919,
    "LastProcessingResult": "OK",
    "State": "Enabled",
    "StateTransitionReason": "User action"
}
```

Untuk menghapus pemetaan sumber peristiwa Amazon DocumentDB Anda, gunakan perintah. `delete-event-source-mapping`

```
aws lambda delete-event-source-mapping \
    --uuid 2b733gdc-8ac3-cdf5-af3a-1827b3b11284
```

Posisi awal polling dan streaming

Ketahuilah bahwa polling streaming selama pembuatan dan pembaruan pemetaan sumber acara pada akhirnya konsisten.

- Selama pembuatan pemetaan sumber acara, mungkin diperlukan beberapa menit untuk memulai acara polling dari aliran.
- Selama pembaruan pemetaan sumber acara, mungkin diperlukan beberapa menit untuk menghentikan dan memulai kembali acara pemungutan suara dari aliran.

Perilaku ini berarti bahwa jika Anda menentukan `LATEST` sebagai posisi awal untuk aliran, pemetaan sumber peristiwa dapat melewatkan peristiwa selama pembuatan atau pembaruan. Untuk memastikan bahwa tidak ada peristiwa yang terlewatkan, tentukan posisi awal aliran sebagai `TRIM_HORIZON` atau`AT_TIMESTAMP`.

Memantau sumber acara Amazon DocumentDB Anda

Untuk membantu Anda memantau sumber peristiwa Amazon DocumentDB, Lambda memancarkan metrik saat fungsi Anda `IteratorAge` selesai memproses sekumpulan rekaman. Usia iterator adalah perbedaan antara stempel waktu acara terbaru dan stempel waktu saat ini. Pada dasarnya, `IteratorAge` metrik menunjukkan berapa umur catatan terakhir yang diproses dalam batch.

Jika fungsi Anda saat ini memproses peristiwa baru, maka Anda dapat menggunakan usia iterator untuk memperkirakan latensi antara saat catatan ditambahkan dan saat fungsi Anda memprosesnya. Tren yang meningkat `IteratorAge` dapat menunjukkan masalah dengan fungsi Anda. Untuk informasi selengkapnya, lihat Menggunakan CloudWatch metrik dengan Lambda.

Aliran perubahan Amazon DocumentDB tidak dioptimalkan untuk menangani kesenjangan waktu yang besar antar peristiwa. Jika sumber peristiwa Amazon DocumentDB Anda tidak menerima peristiwa apa pun untuk jangka waktu yang lama, Lambda mungkin menonaktifkan pemetaan sumber peristiwa. Panjang periode waktu ini dapat bervariasi dari beberapa minggu hingga beberapa bulan tergantung pada ukuran cluster dan beban kerja lainnya.

Lambda mendukung muatan hingga 6 MB. Namun, peristiwa aliran perubahan Amazon DocumentDB dapat berukuran hingga 16 MB. Jika aliran perubahan Anda mencoba mengirim Lambda peristiwa aliran perubahan yang lebih besar dari 6 MB, maka Lambda akan menghapus pesan dan memancarkan metrik. `OversizedRecordCount` Lambda memancarkan semua metrik dengan upaya terbaik.

Tutorial: Menggunakan AWS Lambda dengan Amazon DocumentDB Streams

Dalam tutorial ini, Anda membuat Fungsi Event Lambda dasar yang menggunakan peristiwa dari aliran perubahan Amazon DocumentDB (dengan kompatibilitas MongoDB). Untuk menyelesaikan tutorial ini, Anda melalui tahapan berikut:

- Siapkan cluster Amazon DocumentDB Anda, sambungkan ke sana, dan aktifkan aliran perubahan di dalamnya.
- Buat Fungsi Acara Lambda Anda, dan konfigurasikan cluster Amazon DocumentDB Anda sebagai sumber peristiwa untuk fungsi Anda.
- Uji penyiapan dengan memasukkan item ke dalam database Amazon DocumentDB Anda.

Buat cluster Amazon DocumentDB

1. Buka konsol Amazon DocumentDB. Di bawah Cluster, pilih Buat.
2. Buat cluster dengan konfigurasi berikut:
   - Untuk tipe Cluster, pilih Instance-based cluster. Ini adalah pilihan default.
   - Di bawah konfigurasi Cluster, pastikan bahwa Engine versi 5.0.0 dipilih. Ini adalah pilihan default.
   - Di bawah konfigurasi Instance:
     - Untuk kelas instans DB, pilih kelas yang dioptimalkan memori. Ini adalah pilihan default.
     - Untuk Jumlah contoh replika biasa, pilih 1.
     - Untuk kelas Instance, gunakan pilihan default.
   - Di bawah Autentikasi, masukkan nama pengguna untuk pengguna utama, lalu pilih Dikelola sendiri. Masukkan kata sandi, lalu konfirmasikan.
   - Simpan semua pengaturan default lainnya.
3. Pilih Buat klaster.

Buat rahasia di Secrets Manager

Saat Amazon DocumentDB membuat klaster Anda, buat rahasia untuk menyimpan AWS Secrets Manager kredensi database Anda. Anda akan memberikan rahasia ini saat membuat pemetaan sumber peristiwa Lambda di langkah selanjutnya.

Untuk membuat rahasia di Secrets Manager

1. Buka konsol Secrets Manager dan pilih Simpan rahasia baru.
2. Untuk Pilih jenis rahasia, pilih opsi berikut:
   - Di bawah rincian dasar:
     - Jenis rahasia: Kredensi untuk database Amazon DocumentDB Anda

- Di bawah Kredensial, masukkan nama pengguna dan kata sandi yang sama dengan yang Anda gunakan untuk membuat klaster Amazon DocumentDB Anda.
- Database: Pilih cluster Amazon DocumentDB Anda.
- Pilih Berikutnya.

3. Untuk Konfigurasi rahasia, pilih opsi berikut:
   - Nama rahasia: `DocumentDBSecret`
   - Pilih Berikutnya.
4. Pilih Berikutnya.
5. Pilih Toko.
6. Segarkan konsol untuk memverifikasi bahwa Anda berhasil menyimpan `DocumentDBSecret` rahasia.

Perhatikan Rahasia ARN. Anda akan membutuhkannya di langkah selanjutnya.

Menyambungkan ke klaster

Connect ke cluster Amazon DocumentDB Anda menggunakan AWS CloudShell

1. Di konsol manajemen Amazon DocumentDB, di bawah Clusters, temukan klaster yang Anda buat. Pilih klaster Anda dengan memilih kotak centang di sebelahnya.
2. Pilih Connect to cluster. Layar perintah CloudShell Jalankan muncul.
3. Di bidang Nama lingkungan baru, masukkan nama unik, seperti “uji” dan pilih Buat dan jalankan.
4. Saat diminta, masukkan kata sandi Anda. Ketika prompt menjadi`rs0 [direct: primary] <env-name>>`, Anda berhasil terhubung ke cluster Amazon DocumentDB Anda.

Aktifkan aliran perubahan

Untuk tutorial ini, Anda akan melacak perubahan pada `products` koleksi `docdbdemo` database di cluster Amazon DocumentDB Anda. Anda melakukan ini dengan mengaktifkan aliran perubahan.

Untuk membuat database baru di dalam klaster Anda

1. Jalankan perintah berikut untuk membuat database baru yang disebut`docdbdemo`:

```
use docdbdemo
```

2. Di jendela terminal, gunakan perintah berikut untuk memasukkan catatan ke`docdbdemo`:

```
db.products.insertOne({"hello":"world"})
```

   Anda akan melihat output seperti ini:

```
{
  acknowledged: true,
  insertedId: ObjectId('67f85066ca526410fd531d59')
}
```

3. Selanjutnya, aktifkan aliran perubahan pada `products` koleksi `docdbdemo` database menggunakan perintah berikut:

```
db.adminCommand({modifyChangeStreams: 1,
    database: "docdbdemo",
    collection: "products",
    enable: true});
```

   Anda akan melihat output seperti ini:

```
{ "ok" : 1, "operationTime" : Timestamp(1680126165, 1) }
```

Buat antarmuka VPC endpoint

Selanjutnya, buat titik akhir VPC antarmuka untuk memastikan bahwa Lambda dan Secrets Manager (digunakan nanti untuk menyimpan kredenal akses cluster kami) dapat terhubung ke VPC default Anda.

Untuk membuat titik akhir VPC antarmuka

1. Buka konsol VPC. Di menu sebelah kiri, di bawah Virtual Private Cloud, pilih Endpoints.
2. Pilih Buat titik akhir. Buat titik akhir dengan konfigurasi berikut:
   - Untuk tag Nama, masukkan`lambda-default-vpc`.
   - Untuk kategori Layanan, pilih AWS layanan.
   - Untuk Layanan, masukkan `lambda` di kotak pencarian. Pilih layanan dengan format`com.amazonaws.<region>.lambda`.

- Untuk VPC, pilih VPC tempat klaster Amazon DocumentDB Anda berada. Ini biasanya VPC default.
- Untuk Subnet, centang kotak di sebelah setiap zona ketersediaan. Pilih subnet ID yang benar untuk setiap zona ketersediaan.
- Untuk jenis alamat IP, pilih IPv4.
- Untuk grup Keamanan, pilih grup keamanan yang digunakan cluster Amazon DocumentDB Anda. Ini biasanya kelompok `default` keamanan.
- Simpan semua pengaturan default lainnya.
- Pilih Buat titik akhir.

3. Sekali lagi, pilih Buat titik akhir. Buat titik akhir dengan konfigurasi berikut:

- Untuk tag Nama, masukkan`secretsmanager-default-vpc`.
- Untuk kategori Layanan, pilih AWS layanan.
- Untuk Layanan, masukkan `secretsmanager` di kotak pencarian. Pilih layanan dengan format`com.amazonaws.<region>.secretsmanager`.
- Untuk VPC, pilih VPC tempat klaster Amazon DocumentDB Anda berada. Ini biasanya VPC default.
- Untuk Subnet, centang kotak di sebelah setiap zona ketersediaan. Pilih subnet ID yang benar untuk setiap zona ketersediaan.
- Untuk jenis alamat IP, pilih IPv4.
- Untuk grup Keamanan, pilih grup keamanan yang digunakan cluster Amazon DocumentDB Anda. Ini biasanya kelompok `default` keamanan.
- Simpan semua pengaturan default lainnya.
- Pilih Buat titik akhir.

Ini melengkapi bagian pengaturan cluster dari tutorial ini.

Buat peran eksekusi

Pada rangkaian langkah berikutnya, Anda akan membuat Fungsi Acara Lambda Anda. Pertama, Anda perlu membuat peran eksekusi yang memberikan izin fungsi Anda untuk mengakses klaster Anda. Anda melakukannya dengan membuat kebijakan IAM terlebih dahulu, lalu melampirkan kebijakan ini ke peran IAM.

Untuk membuat kebijakan IAM

1. Buka halaman Kebijakan di konsol IAM dan pilih Buat kebijakan.
2. Pilih tab JSON. Dalam kebijakan berikut, ganti ARN sumber daya Secrets Manager di baris terakhir pernyataan dengan ARN rahasia Anda dari sebelumnya, dan salin kebijakan ke editor.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "LambdaESMNetworkingAccess",
            "Effect": "Allow",
            "Action": [
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DescribeVpcs",
                "ec2:DeleteNetworkInterface",
                "ec2:DescribeSubnets",
                "ec2:DescribeSecurityGroups",
                "kms:Decrypt"
            ],
            "Resource": "*"
        },
        {
            "Sid": "LambdaDocDBESMAccess",
            "Effect": "Allow",
            "Action": [
                "rds:DescribeDBClusters",
                "rds:DescribeDBClusterParameters",
                "rds:DescribeDBSubnetGroups"
            ],
            "Resource": "*"
        },
        {
            "Sid": "LambdaDocDBESMGetSecretValueAccess",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue"
            ],
```

```
            "Resource": "arn:aws:secretsmanager:us-
east-1:123456789012:secret:DocumentDBSecret"
        }
    ]
}
```

3. Pilih Berikutnya: Tag, lalu pilih Berikutnya: Tinjau.
4. Untuk Nama, masukkan `AWSDocumentDBLambdaPolicy`.
5. Pilih Buat kebijakan.

Untuk membuat peran IAM

1. Buka halaman Peran di konsol IAM dan pilih Buat peran.
2. Untuk Pilih entitas tepercaya, pilih opsi berikut:
   - Jenis entitas tepercaya: AWS layanan
   - Kasus layanan atau penggunaan: Lambda
   - Pilih Berikutnya.
3. Untuk Menambahkan izin, pilih `AWSDocumentDBLambdaPolicy` kebijakan yang baru saja Anda buat, serta `AWSLambdaBasicExecutionRole` untuk memberikan izin fungsi Anda untuk menulis ke Amazon CloudWatch Logs.
4. Pilih Berikutnya.
5. Untuk Nama peran, masukkan `AWSDocumentDBLambdaExecutionRole`.
6. Pilih Buat peran.

Buat Fungsi Acara Lambda

Tutorial ini menggunakan runtime Python 3.14, tetapi kami juga menyediakan contoh file kode untuk runtime lainnya. Anda dapat memilih tab di kotak berikut untuk melihat kode runtime yang Anda minati.

Kode menerima input peristiwa Amazon DocumentDB dan memproses pesan yang dikandungnya.

Untuk membuat Fungsi Acara Lambda

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih Buat Fungsi Acara.

3. Pilih Penulis dari awal
4. Di bagian Informasi dasar, lakukan hal berikut:
   a. Untuk nama Fungsi, masukkan `ProcessDocumentDBRecords`
   b. Untuk Runtime, pilih Python 3.14.
   c. Untuk Arsitektur, pilih x86_64.
5. Di tab Ubah peran eksekusi default, lakukan hal berikut:
   a. Perluas tab, lalu pilih Gunakan peran yang ada.
   b. Pilih yang `AWSDocumentDBLambdaExecutionRole` Anda buat sebelumnya.
6. Pilih Buat Fungsi Acara.

Untuk menyebarkan kode Fungsi Acara

1. Pilih tab Python di kotak berikut dan salin kodenya.

   .NET

   SDK untuk .NET

   > **Note**
   >
   > Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

   Mengkonsumsi acara Amazon DocumentDB dengan Lambda menggunakan.NET.

```
using Amazon.Lambda.Core;
using System.Text.Json;
using System;
using System.Collections.Generic;
using System.Text.Json.Serialization;
//Assembly attribute to enable the Lambda function's JSON input to be
 converted into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJson

namespace LambdaDocDb;
```

```
public class Function
{

     /// <summary>
    /// Lambda function entry point to process Amazon DocumentDB events.
    /// </summary>
    /// <param name="event">The Amazon DocumentDB event.</param>
    /// <param name="context">The Lambda context object.</param>
    /// <returns>A string to indicate successful processing.</returns>
    public string FunctionHandler(Event evnt, ILambdaContext context)
    {

        foreach (var record in evnt.Events)
        {
            ProcessDocumentDBEvent(record, context);
        }

        return "OK";
    }

     private void ProcessDocumentDBEvent(DocumentDBEventRecord record,
 ILambdaContext context)
    {

        var eventData = record.Event;
        var operationType = eventData.OperationType;
        var databaseName = eventData.Ns.Db;
        var collectionName = eventData.Ns.Coll;
        var fullDocument = JsonSerializer.Serialize(eventData.FullDocument,
 new JsonSerializerOptions { WriteIndented = true });

        context.Logger.LogLine($"Operation type: {operationType}");
        context.Logger.LogLine($"Database: {databaseName}");
        context.Logger.LogLine($"Collection: {collectionName}");
        context.Logger.LogLine($"Full document:\n{fullDocument}");
    }



    public class Event
    {
        [JsonPropertyName("eventSourceArn")]
        public string EventSourceArn { get; set; }
```

```
        [JsonPropertyName("events")]
        public List<DocumentDBEventRecord> Events { get; set; }

        [JsonPropertyName("eventSource")]
        public string EventSource { get; set; }
    }

    public class DocumentDBEventRecord
    {
        [JsonPropertyName("event")]
        public EventData Event { get; set; }
    }

    public class EventData
    {
        [JsonPropertyName("_id")]
        public IdData Id { get; set; }

        [JsonPropertyName("clusterTime")]
        public ClusterTime ClusterTime { get; set; }

        [JsonPropertyName("documentKey")]
        public DocumentKey DocumentKey { get; set; }

        [JsonPropertyName("fullDocument")]
        public Dictionary<string, object> FullDocument { get; set; }

        [JsonPropertyName("ns")]
        public Namespace Ns { get; set; }

        [JsonPropertyName("operationType")]
        public string OperationType { get; set; }
    }

    public class IdData
    {
        [JsonPropertyName("_data")]
        public string Data { get; set; }
    }

    public class ClusterTime
    {
        [JsonPropertyName("$timestamp")]
```

```
        public Timestamp Timestamp { get; set; }
    }

    public class Timestamp
    {
        [JsonPropertyName("t")]
        public long T { get; set; }

        [JsonPropertyName("i")]
        public int I { get; set; }
    }

    public class DocumentKey
    {
        [JsonPropertyName("_id")]
        public Id Id { get; set; }
    }

    public class Id
    {
        [JsonPropertyName("$oid")]
        public string Oid { get; set; }
    }

    public class Namespace
    {
        [JsonPropertyName("db")]
        public string Db { get; set; }

        [JsonPropertyName("coll")]
        public string Coll { get; set; }
    }
}
```

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara Amazon DocumentDB dengan Lambda menggunakan Go.

```
package main

import (
 "context"
 "encoding/json"
 "fmt"

 "github.com/aws/aws-lambda-go/lambda"
)

type Event struct {
 Events []Record `json:"events"`
}

type Record struct {
 Event struct {
  OperationType string `json:"operationType"`
  NS            struct {
   DB   string `json:"db"`
   Coll string `json:"coll"`
  } `json:"ns"`
  FullDocument interface{} `json:"fullDocument"`
 } `json:"event"`
}

func main() {
 lambda.Start(handler)
}
```

```
func handler(ctx context.Context, event Event) (string, error) {
 fmt.Println("Loading function")
 for _, record := range event.Events {
  logDocumentDBEvent(record)
 }

 return "OK", nil
}

func logDocumentDBEvent(record Record) {
 fmt.Printf("Operation type: %s\n", record.Event.OperationType)
 fmt.Printf("db: %s\n", record.Event.NS.DB)
 fmt.Printf("collection: %s\n", record.Event.NS.Coll)
 docBytes, _ := json.MarshalIndent(record.Event.FullDocument, "", "  ")
 fmt.Printf("Full document: %s\n", string(docBytes))
}
```

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara Amazon DocumentDB dengan Lambda menggunakan Java.

```
import java.util.List;
import java.util.Map;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;

public class Example implements RequestHandler<Map<String, Object>, String> {

    @SuppressWarnings("unchecked")
    @Override
```

```
    public String handleRequest(Map<String, Object> event, Context context) {
        List<Map<String, Object>> events = (List<Map<String, Object>>)
 event.get("events");
        for (Map<String, Object> record : events) {
            Map<String, Object> eventData = (Map<String, Object>)
 record.get("event");
            processEventData(eventData);
        }

        return "OK";
    }

    @SuppressWarnings("unchecked")
    private void processEventData(Map<String, Object> eventData) {
        String operationType = (String) eventData.get("operationType");
        System.out.println("operationType: %s".formatted(operationType));

        Map<String, Object> ns = (Map<String, Object>) eventData.get("ns");

        String db = (String) ns.get("db");
        System.out.println("db: %s".formatted(db));
        String coll = (String) ns.get("coll");
        System.out.println("coll: %s".formatted(coll));

        Map<String, Object> fullDocument = (Map<String, Object>)
 eventData.get("fullDocument");
        System.out.println("fullDocument: %s".formatted(fullDocument));
    }

}
```

JavaScript

SDK untuk JavaScript (v3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara Amazon DocumentDB dengan menggunakan Lambda. JavaScript

```
console.log('Loading function');
exports.handler = async (event, context) => {
    event.events.forEach(record => {
        logDocumentDBEvent(record);
    });
    return 'OK';
};

const logDocumentDBEvent = (record) => {
    console.log('Operation type: ' + record.event.operationType);
    console.log('db: ' + record.event.ns.db);
    console.log('collection: ' + record.event.ns.coll);
    console.log('Full document:', JSON.stringify(record.event.fullDocument,
 null, 2));
};
```

Mengkonsumsi acara Amazon DocumentDB dengan Lambda menggunakan TypeScript

```
import { DocumentDBEventRecord, DocumentDBEventSubscriptionContext } from
 'aws-lambda';

console.log('Loading function');

export const handler = async (
  event: DocumentDBEventSubscriptionContext,
  context: any
): Promise<string> => {
  event.events.forEach((record: DocumentDBEventRecord) => {
    logDocumentDBEvent(record);
  });
  return 'OK';
};

const logDocumentDBEvent = (record: DocumentDBEventRecord): void => {
  console.log('Operation type: ' + record.event.operationType);
  console.log('db: ' + record.event.ns.db);
  console.log('collection: ' + record.event.ns.coll);
```

```
  console.log('Full document:', JSON.stringify(record.event.fullDocument,
 null, 2));
};
```

PHP

SDK untuk PHP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara Amazon DocumentDB dengan Lambda menggunakan PHP.

```
<?php

require __DIR__.'/vendor/autoload.php';

use Bref\Context\Context;
use Bref\Event\Handler;

class DocumentDBEventHandler implements Handler
{
    public function handle($event, Context $context): string
    {

        $events = $event['events'] ?? [];
        foreach ($events as $record) {
            $this->logDocumentDBEvent($record['event']);
        }
        return 'OK';
    }

    private function logDocumentDBEvent($event): void
    {
        // Extract information from the event record

        $operationType = $event['operationType'] ?? 'Unknown';
```

```
        $db = $event['ns']['db'] ?? 'Unknown';
        $collection = $event['ns']['coll'] ?? 'Unknown';
        $fullDocument = $event['fullDocument'] ?? [];

        // Log the event details

        echo "Operation type: $operationType\n";
        echo "Database: $db\n";
        echo "Collection: $collection\n";
        echo "Full document: " . json_encode($fullDocument,
 JSON_PRETTY_PRINT) . "\n";
    }
}
return new DocumentDBEventHandler();
```

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara Amazon DocumentDB dengan Lambda menggunakan Python.

```
import json

def lambda_handler(event, context):
    for record in event.get('events', []):
        log_document_db_event(record)
    return 'OK'

def log_document_db_event(record):
    event_data = record.get('event', {})
    operation_type = event_data.get('operationType', 'Unknown')
    db = event_data.get('ns', {}).get('db', 'Unknown')
    collection = event_data.get('ns', {}).get('coll', 'Unknown')
    full_document = event_data.get('fullDocument', {})
```

```
    print(f"Operation type: {operation_type}")
    print(f"db: {db}")
    print(f"collection: {collection}")
    print("Full document:", json.dumps(full_document, indent=2))
```

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara Amazon DocumentDB dengan Lambda menggunakan Ruby.

```
require 'json'

def lambda_handler(event:, context:)
  event['events'].each do |record|
    log_document_db_event(record)
  end
  'OK'
end

def log_document_db_event(record)
  event_data = record['event'] || {}
  operation_type = event_data['operationType'] || 'Unknown'
  db = event_data.dig('ns', 'db') || 'Unknown'
  collection = event_data.dig('ns', 'coll') || 'Unknown'
  full_document = event_data['fullDocument'] || {}

  puts "Operation type: #{operation_type}"
  puts "db: #{db}"
  puts "collection: #{collection}"
  puts "Full document: #{JSON.pretty_generate(full_document)}"
end
```

Rust

SDK for Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara Amazon DocumentDB dengan Lambda menggunakan Rust.

```
use lambda_runtime::{service_fn, tracing, Error, LambdaEvent};
use aws_lambda_events::{
    event::documentdb::{DocumentDbEvent, DocumentDbInnerEvent},
   };


// Built with the following dependencies:
//lambda_runtime = "0.11.1"
//serde_json = "1.0"
//tokio = { version = "1", features = ["macros"] }
//tracing = { version = "0.1", features = ["log"] }
//tracing-subscriber = { version = "0.3", default-features = false, features
 = ["fmt"] }
//aws_lambda_events = "0.15.0"

async fn function_handler(event: LambdaEvent<DocumentDbEvent>) ->Result<(),
 Error> {

    tracing::info!("Event Source ARN: {:?}", event.payload.event_source_arn);
    tracing::info!("Event Source: {:?}", event.payload.event_source);

    let records = &event.payload.events;

    if records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(());
    }
```

```
    for record in records{
        log_document_db_event(record);
    }

    tracing::info!("Document db records processed");

    // Prepare the response
    Ok(())

}

fn log_document_db_event(record: &DocumentDbInnerEvent)-> Result<(), Error>{
    tracing::info!("Change Event: {:?}", record.event);

    Ok(())

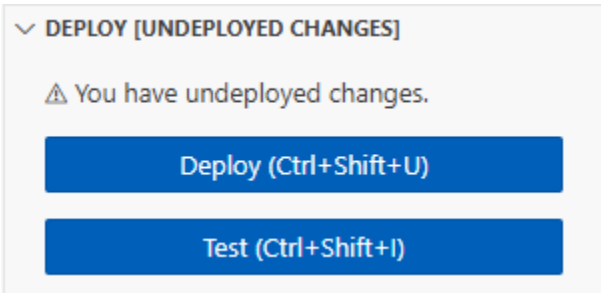
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
    .with_max_level(tracing::Level::INFO)
    .with_target(false)
    .without_time()
    .init();

    let func = service_fn(function_handler);
    lambda_runtime::run(func).await?;
    Ok(())

}
```

2. Di panel Sumber kode di konsol Lambda, tempelkan kode ke editor kode, ganti kode yang dibuat Lambda.
3. Di bagian DEPLOY, pilih Deploy untuk memperbarui kode fungsi Anda:



Buat pemetaan sumber acara Lambda

Buat pemetaan sumber peristiwa yang mengaitkan aliran perubahan Amazon DocumentDB Anda dengan Fungsi Acara Lambda Anda. Setelah Anda membuat pemetaan sumber acara ini, AWS Lambda segera mulai polling aliran.

Untuk membuat pemetaan sumber acara

1. Buka halaman Fungsi Acara di konsol Lambda.
2. Pilih `ProcessDocumentDBRecords` fungsi yang Anda buat sebelumnya.
3. Pilih tab Konfigurasi, lalu pilih Pemicu di menu sebelah kiri.
4. Pilih Tambahkan pemicu.
5. Di bawah konfigurasi Trigger, untuk sumbernya, pilih Amazon DocumentDB.
6. Buat pemetaan sumber acara dengan konfigurasi berikut:
   - Cluster Amazon DocumentDB: Pilih cluster yang Anda buat sebelumnya.
   - Nama database: docdbdemo
   - Nama koleksi: produk
   - Ukuran Batch: 1
   - Posisi awal: Terbaru
   - Otentikasi: BASIC_AUTH
   - Kunci Secrets Manager: Pilih rahasia untuk cluster Amazon DocumentDB Anda. Itu disebut sesuatu seperti`rds!cluster-12345678-a6f0-52c0-b290-db4aga89274f`.
   - Jendela Batch: 1
   - Konfigurasi dokumen lengkap: UpdateLookup
7. Pilih Tambahkan. Membuat pemetaan sumber acara Anda dapat memakan waktu beberapa menit.

Uji Fungsi Acara Anda

Tunggu pemetaan sumber peristiwa mencapai status Diaktifkan. Ini dapat memakan waktu beberapa menit. Kemudian, uji penyiapan ujung ke ujung dengan memasukkan, memperbarui, dan menghapus catatan basis data. Sebelum Anda memulai:

1. Sambungkan kembali ke cluster Amazon DocumentDB di lingkungan Anda. CloudShell
2. Jalankan perintah berikut untuk memastikan bahwa Anda menggunakan `docdbdemo` database:

```
use docdbdemo
```

Masukkan catatan

Masukkan catatan ke dalam `products` koleksi `docdbdemo` database:

```
db.products.insertOne({"name":"Pencil", "price": 1.00})
```

Verifikasi bahwa fungsi Anda berhasil memproses peristiwa ini dengan memeriksa CloudWatch Log. Anda akan melihat entri log seperti ini:

Timestamp	Message
2025-05-05T23:55:23.474Z	Operation type: insert
2025-05-05T23:55:23.474Z	db: docdbdemo
2025-05-05T23:55:23.474Z	collection: products
2025-05-05T23:55:23.474Z	Full document: {
2025-05-05T23:55:23.474Z	"_id": {
2025-05-05T23:55:23.474Z	"$oid": "68194fea08d12462ea531d58"
2025-05-05T23:55:23.474Z	},
2025-05-05T23:55:23.474Z	"name": "Pencil",
2025-05-05T23:55:23.474Z	"price": 1
2025-05-05T23:55:23.474Z	}

Perbarui catatan

Perbarui catatan yang baru saja Anda masukkan dengan perintah berikut:

```
db.products.updateOne(
    { "name": "Pencil" },
    { $set: { "price": 0.50 }}
)
```

Verifikasi bahwa fungsi Anda berhasil memproses peristiwa ini dengan memeriksa CloudWatch Log. Anda akan melihat entri log seperti ini:

Timestamp	Message
2025-05-05T23:55:27.918Z	Operation type: update
2025-05-05T23:55:27.918Z	db: docdbdemo
2025-05-05T23:55:27.918Z	collection: products
2025-05-05T23:55:27.918Z	Full document: {
2025-05-05T23:55:27.918Z	"_id": {
2025-05-05T23:55:27.918Z	"$oid": "68194fea08d12462ea531d58"
2025-05-05T23:55:27.918Z	},
2025-05-05T23:55:27.918Z	"name": "Pencil",
2025-05-05T23:55:27.918Z	"price": 0.5
2025-05-05T23:55:27.918Z	}

Hapus catatan

Hapus catatan yang baru saja Anda perbarui dengan perintah berikut:

```
db.products.deleteOne( { "name": "Pencil" } )
```

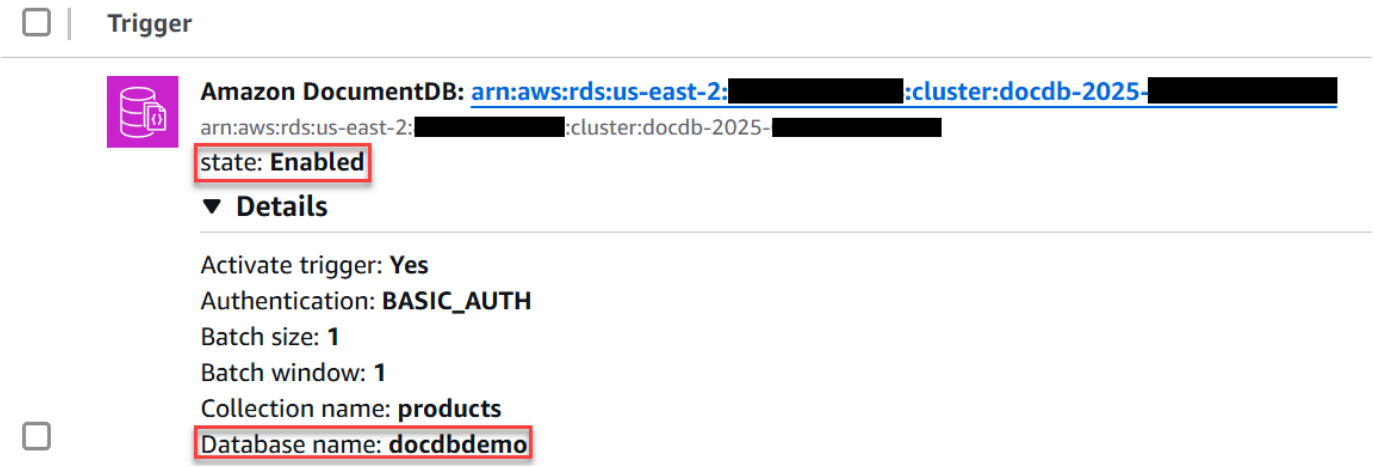
Verifikasi bahwa fungsi Anda berhasil memproses peristiwa ini dengan memeriksa CloudWatch Log. Anda akan melihat entri log seperti ini:

Timestamp	Message
2025-05-05T23:55:32.362Z	Operation type: delete
2025-05-05T23:55:32.362Z	db: docdbdemo
2025-05-05T23:55:32.362Z	collection: products
2025-05-05T23:55:32.362Z	Full document: {}

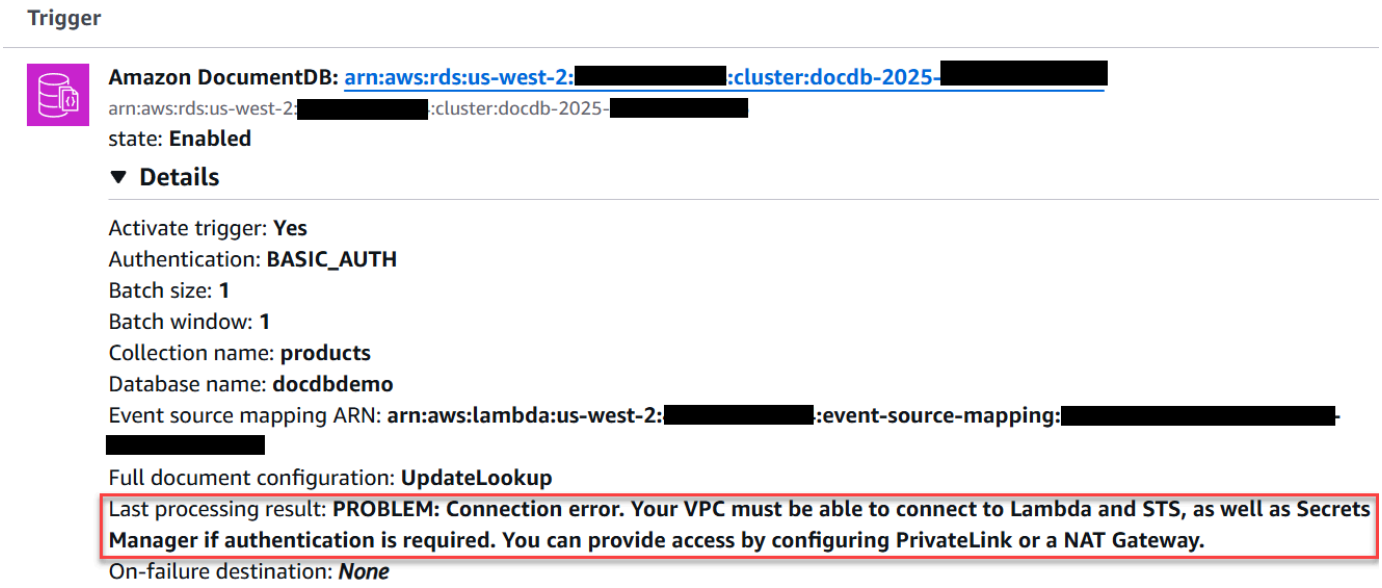
Pemecahan masalah

Jika Anda tidak melihat peristiwa database apa pun di CloudWatch log fungsi Anda, periksa hal berikut:

- Pastikan bahwa pemetaan sumber peristiwa Lambda (juga dikenal sebagai pemicu) dalam status Diaktifkan. Pemetaan sumber acara dapat memakan waktu beberapa menit untuk dibuat.
- Jika pemetaan sumber peristiwa Diaktifkan tetapi Anda masih tidak melihat peristiwa database di CloudWatch:
  - Pastikan bahwa nama Database dalam pemetaan sumber peristiwa diatur ke`docdbdemo`.



  - Periksa bidang pemetaan sumber peristiwa Hasil pemrosesan terakhir untuk pesan berikut “MASALAH: Kesalahan koneksi. VPC Anda harus dapat terhubung ke Lambda dan STS, serta Secrets Manager jika diperlukan otentikasi. Jika Anda melihat kesalahan ini, pastikan Anda membuat titik akhir antarmuka VPC Lambda dan Secrets Manager, dan titik akhir menggunakan VPC dan subnet yang sama dengan yang digunakan cluster Amazon DocumentDB Anda.



Bersihkan sumber daya Anda

Sekarang Anda dapat menghapus sumber daya yang Anda buat untuk tutorial ini, kecuali Anda ingin mempertahankannya. Dengan menghapus sumber daya AWS yang tidak lagi Anda gunakan, Anda mencegah biaya yang tidak perlu untuk Akun AWS Anda.

Untuk menghapus Fungsi Acara Lambda

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih Fungsi Acara yang Anda buat.
3. Pilih Tindakan, Hapus.
4. Ketik **`confirm`** kolom input teks dan pilih Hapus.

Untuk menghapus peran eksekusi

1. Buka halaman Peran dari konsol IAM.
2. Pilih peran eksekusi yang Anda buat.
3. Pilih Hapus.
4. Masukkan nama peran di bidang input teks dan pilih Hapus.

Untuk menghapus titik akhir VPC

1. Buka konsol VPC. Di menu sebelah kiri, di bawah Virtual Private Cloud, pilih Endpoints.
2. Pilih titik akhir yang Anda buat.
3. Pilih Tindakan, Hapus titik akhir VPC.
4. Masukkan **`delete`** di bidang input teks.
5. Pilih Hapus.

Untuk menghapus cluster Amazon DocumentDB

1. Buka konsol Amazon DocumentDB.
2. Pilih cluster Amazon DocumentDB yang Anda buat untuk tutorial ini, dan nonaktifkan perlindungan penghapusan.
3. Di halaman utama Clusters, pilih kembali cluster Amazon DocumentDB Anda.
4. Pilih Tindakan, Hapus.
5. Untuk Buat snapshot klaster akhir, pilih No.
6. Masukkan **`delete`** di bidang input teks.
7. Pilih Hapus.

Untuk menghapus rahasia di Secrets Manager

1. Buka konsol Secrets Manager.
2. Pilih rahasia yang Anda buat untuk tutorial ini.
3. Pilih Tindakan, Hapus rahasia.
4. Pilih Jadwalkan penghapusan.

Penggunaan AWS Lambda dengan Amazon DynamoDB

> **Note**
>
> Jika Anda ingin mengirim data ke target selain Fungsi Peristiwa Lambda atau memperkaya data sebelum mengirimnya, lihat Amazon Pipes. EventBridge

Anda dapat menggunakan Fungsi AWS Lambda Peristiwa untuk memproses rekaman di aliran Amazon DynamoDB. Dengan DynamoDB Streams, Anda dapat memicu Fungsi Peristiwa Lambda untuk melakukan pekerjaan tambahan setiap kali tabel DynamoDB diperbarui.

Saat memproses aliran DynamoDB, Anda perlu menerapkan logika respons batch sebagian untuk mencegah catatan yang berhasil diproses dicoba ulang ketika beberapa catatan dalam batch gagal. Utilitas Batch Processor dari Powertools for AWS Lambda tersedia dalam Python TypeScript,, .NET, dan Java dan menyederhanakan implementasi ini dengan secara otomatis menangani logika respons batch sebagian, mengurangi waktu pengembangan dan meningkatkan keandalan.

Topik

- Polling dan batching stream
- Posisi awal polling dan streaming
- Pembaca serpihan secara bersamaan di DynamoDB Streams
- Contoh peristiwa
- Memproses catatan DynamoDB dengan Lambda
- Mengkonfigurasi respons batch sebagian dengan DynamoDB dan Lambda
- Menyimpan catatan yang dibuang untuk sumber peristiwa DynamoDB di Lambda
- Menerapkan pemrosesan aliran DynamoDB stateful di Lambda
- Parameter Lambda untuk pemetaan sumber peristiwa Amazon DynamoDB
- Menggunakan pemfilteran acara dengan sumber acara DynamoDB
- Tutorial: Menggunakan AWS Lambda dengan aliran Amazon DynamoDB

Polling dan batching stream

Lambda melakukan polling shard dalam aliran DynamoDB untuk catatan dengan tingkat dasar sebanyak 4 kali per detik. Saat rekaman tersedia, Lambda memanggil fungsi Anda dan menunggu

hasilnya. Jika pemrosesan berhasil, Lambda melanjutkan polling sampai menerima lebih banyak catatan.

Secara default, Lambda memanggil fungsi Anda segera setelah catatan tersedia. Jika batch yang dibaca Lambda dari sumber peristiwa hanya memiliki satu catatan di dalamnya, Lambda hanya mengirimkan satu catatan ke fungsi tersebut. Untuk menghindari menjalankan fungsi dengan sejumlah kecil catatan, Anda dapat memberi tahu sumber acara untuk menyangga catatan hingga 5 menit dengan mengonfigurasi jendela batching. Sebelum menjalankan fungsi, Lambda terus membaca catatan dari sumber acara hingga mengumpulkan batch penuh, jendela batching kedaluwarsa, atau batch mencapai batas muatan 6 MB. Untuk informasi selengkapnya, lihat Perilaku batching.

> **Warning**
>
> Pemetaan sumber peristiwa Lambda memproses setiap peristiwa setidaknya sekali, dan pemrosesan duplikat catatan dapat terjadi. Untuk menghindari potensi masalah yang terkait dengan duplikat peristiwa, kami sangat menyarankan agar Anda membuat kode fungsi Anda idempoten. Untuk mempelajari lebih lanjut, lihat Bagaimana cara membuat fungsi Lambda saya idempoten di Pusat Pengetahuan. AWS

Lambda tidak menunggu ekstensi yang dikonfigurasi selesai sebelum mengirim batch berikutnya untuk diproses. Dengan kata lain, ekstensi Anda dapat terus berjalan saat Lambda memproses kumpulan catatan berikutnya. Hal ini dapat menyebabkan masalah pembatasan jika Anda melanggar pengaturan atau batasan konkurensi akun Anda. Untuk mendeteksi apakah ini merupakan masalah potensial, pantau fungsi Anda dan periksa apakah Anda melihat metrik konkurensi yang lebih tinggi dari yang diharapkan untuk pemetaan sumber peristiwa Anda. Karena waktu yang singkat di antara pemanggilan, Lambda mungkin secara singkat melaporkan penggunaan konkurensi yang lebih tinggi daripada jumlah pecahan. Ini bisa benar bahkan untuk Lambda Event Functions tanpa ekstensi.

Konfigurasikan ParallelizationFactorpengaturan untuk memproses satu pecahan aliran DynamoDB dengan lebih dari satu pemanggilan Lambda secara bersamaan. Anda dapat menentukan jumlah batch bersamaan yang dipolling Lambda dari pecahan dengan menggunakan faktor paralelisasi dari 1 (default) hingga 10. Misalnya, saat Anda menyetel `ParallelizationFactor` ke 2, Anda dapat memiliki maksimum 200 pemanggilan Lambda bersamaan untuk memproses 100 pecahan aliran DynamoDB (meskipun dalam praktiknya, Anda mungkin melihat nilai yang berbeda untuk metrik). `ConcurrentExecutions` Hal ini membantu meningkatkan skala throughput pemrosesan ketika volume data tidak stabil dan IteratorAge tinggi. Dengan`ParallelizationFactor`, Lambda

menjalankan beberapa batch bersamaan per pecahan dan mengelompokkan catatan di setiap pecahan dengan kunci partisi. Dalam pecahan, catatan yang berbagi kunci partisi masuk ke batch bersamaan yang sama dan diproses secara berurutan. Catatan dengan kunci partisi yang berbeda dapat diproses secara bersamaan. DynamoDB Streams menjamin pemesanan per item (kunci partisi dan kunci sortir). Akibatnya, catatan aliran untuk item tunggal apa pun selalu muncul dalam urutan item dimodifikasi.

Posisi awal polling dan streaming

Ketahuilah bahwa polling streaming selama pembuatan dan pembaruan pemetaan sumber acara pada akhirnya konsisten.

- Selama pembuatan pemetaan sumber acara, mungkin diperlukan beberapa menit untuk memulai acara polling dari aliran.
- Selama pembaruan pemetaan sumber acara, mungkin diperlukan beberapa menit untuk menghentikan dan memulai kembali acara pemungutan suara dari aliran.

Perilaku ini berarti bahwa jika Anda menentukan `LATEST` sebagai posisi awal untuk aliran, pemetaan sumber peristiwa dapat melewatkan peristiwa selama pembuatan atau pembaruan. Untuk memastikan bahwa tidak ada peristiwa yang terlewatkan, tentukan posisi awal aliran sebagai`TRIM_HORIZON`.

Pembaca serpihan secara bersamaan di DynamoDB Streams

Untuk tabel Single-region yang bukan tabel global, Anda dapat mendesain hingga dua Fungsi Peristiwa Lambda untuk dibaca dari pecahan DynamoDB Streams yang sama secara bersamaan. Melebihi batas ini dapat mengakibatkan throttling permintaan. Untuk tabel global, kami sarankan Anda membatasi jumlah fungsi simultan menjadi satu untuk menghindari pembatasan permintaan.

Contoh peristiwa

Example

```
{
  "Records": [
    {
      "eventID": "1",
      "eventVersion": "1.0",
      "dynamodb": {
        "Keys": {
```

```
          "Id": {
            "N": "101"
          }
        },
        "NewImage": {
          "Message": {
            "S": "New item!"
          },
          "Id": {
            "N": "101"
          }
        },
        "StreamViewType": "NEW_AND_OLD_IMAGES",
        "SequenceNumber": "111",
        "SizeBytes": 26
      },
      "awsRegion": "us-west-2",
      "eventName": "INSERT",
      "eventSourceARN": "arn:aws:dynamodb:us-east-2:123456789012:table/my-table/
stream/2024-06-10T19:26:16.525",
      "eventSource": "aws:dynamodb"
    },
    {
      "eventID": "2",
      "eventVersion": "1.0",
      "dynamodb": {
        "OldImage": {
          "Message": {
            "S": "New item!"
          },
          "Id": {
            "N": "101"
          }
        },
        "SequenceNumber": "222",
        "Keys": {
          "Id": {
            "N": "101"
          }
        },
        "SizeBytes": 59,
        "NewImage": {
          "Message": {
            "S": "This item has changed"
```

```
          },
          "Id": {
            "N": "101"
          }
        },
        "StreamViewType": "NEW_AND_OLD_IMAGES"
      },
      "awsRegion": "us-west-2",
      "eventName": "MODIFY",
      "eventSourceARN": "arn:aws:dynamodb:us-east-2:123456789012:table/my-table/
stream/2024-06-10T19:26:16.525",
      "eventSource": "aws:dynamodb"
    }
  ]}
```

Memproses catatan DynamoDB dengan Lambda

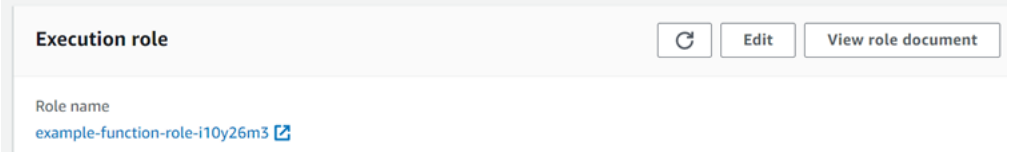
Buat pemetaan sumber peristiwa untuk memberi tahu Lambda mengirim catatan dari aliran Anda ke Fungsi Peristiwa Lambda. Anda dapat membuat beberapa pemetaan sumber peristiwa untuk memproses data yang sama dengan beberapa Fungsi Peristiwa Lambda, atau untuk memproses item dari beberapa aliran dengan satu fungsi.

Anda dapat mengonfigurasi pemetaan sumber peristiwa untuk memproses catatan dari aliran yang berbeda. Akun AWS Untuk mempelajari selengkapnya, lihat the section called “Cross-account pemetaan”.

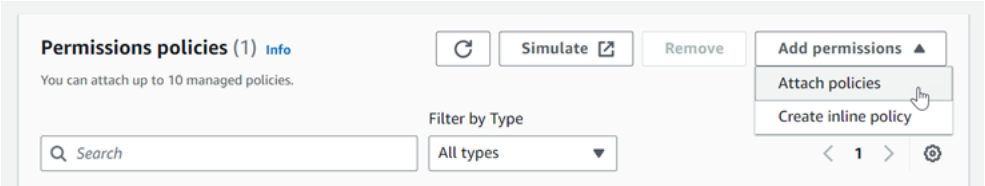
Untuk mengonfigurasi fungsi agar dibaca dari DynamoDB Streams, lampirkan kebijakan terkelola ke peran eksekusi, lalu buat AWSLambdaDynamoDBExecutionRole AWS pemicu DynamoDB.

Untuk menambahkan izin dan membuat pemicu

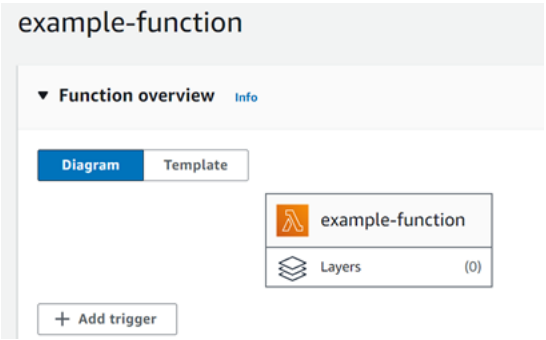
1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih nama sebuah fungsi.
3. Pilih tab Konfigurasi, lalu pilih Izin.
4. Di bawah Nama peran, pilih tautan ke peran eksekusi Anda. Tautan ini membuka peran di konsol IAM.



5. Pilih Tambahkan izin, lalu pilih Lampirkan kebijakan.



6. Di bidang pencarian, masukkan`AWSLambdaDynamoDBExecutionRole`. Tambahkan kebijakan ini ke peran eksekusi Anda. Ini adalah kebijakan AWS terkelola yang berisi izin yang perlu dibaca fungsi Anda dari aliran DynamoDB. Untuk informasi selengkapnya tentang kebijakan ini, lihat AWSLambdaDynamoDBExecutionRoledi Referensi Kebijakan AWS Terkelola.
7. Kembali ke fungsi Anda di konsol Lambda. Di bagian Gambaran umum fungsi, pilih Tambah pemicu.



8. Pilih jenis pemicu.
9. Konfigurasikan opsi yang diperlukan, lalu pilih Tambah.

Lambda mendukung opsi berikut untuk sumber acara DynamoDB:

Opsi sumber kejadian

- Tabel DynamoDB – Tabel DynamoDB tempat untuk membaca catatan.
- Ukuran batch – Jumlah rekaman yang akan dikirimkan ke fungsi dalam setiap batch, paling banyak 10.000. Lambda meneruskan semua rekaman dalam batch ke fungsi dalam satu panggilan, selama ukuran total kejadian tidak melebihi batas payload untuk invokasi sinkron (6 MB).
- Jendela batch – Tentukan jumlah waktu maksimum untuk mengumpulkan rekaman sebelum memanggil fungsi, dalam hitungan detik.
- Posisi mulai – Hanya memproses rekaman baru, atau semua rekaman yang ada.
  - Terbaru – Memproses rekaman baru yang ditambahkan ke stream.
  - Trim horizon – Memproses semua rekaman dalam aliran.

Setelah memproses rekaman yang sudah ada, fungsi berhenti dan melanjutkan memproses rekaman baru.

- On-failure tujuan — Antrian SQS standar atau topik SNS standar untuk catatan yang tidak dapat diproses. Ketika Lambda membuang sekumpulan catatan yang terlalu tua atau telah kehabisan semua percobaan ulang, Lambda mengirimkan detail tentang batch ke antrian atau topik.
- Percobaan ulang – Jumlah maksimum percobaan ulang Lambda saat fungsi memunculkan kesalahan. Hal ini tidak berlaku untuk kesalahan layanan atau pembatasan di mana batch tidak mencapai fungsi.
- Maximum age of record – Usia maksimum rekaman yang dikirimkan Lambda ke fungsi Anda.
- Split batch on error – Ketika fungsi mengembalikan kesalahan, bagi batch menjadi dua bagian sebelum mencoba kembali. Pengaturan ukuran batch asli Anda tetap tidak berubah.
- Batch bersamaan per pecahan — Secara bersamaan memproses beberapa batch dari pecahan yang sama.
- Diaktifkan – Atur ke true untuk mengaktifkan pemetaan sumber kejadian. Atur ke false untuk menghentikan pemrosesan rekaman. Lambda mencatat rekaman terakhir yang diproses dan melanjutkan pemrosesan dari titik tersebut saat pemetaan diaktifkan kembali.

> **Note**
>
> Anda tidak dikenakan biaya untuk panggilan GetRecords API yang dipanggil oleh Lambda sebagai bagian dari pemicu DynamoDB.

Untuk mengelola konfigurasi sumber kejadian nanti, pilih pemicu di desainer.

Membuat pemetaan sumber peristiwa lintas akun

Amazon DynamoDB sekarang mendukung kebijakan berbasis sumber daya. Dengan kemampuan ini, Anda dapat memproses data dari aliran DynamoDB menjadi satu Akun AWS dengan Fungsi Peristiwa Lambda di akun lain.

Untuk membuat pemetaan sumber peristiwa untuk Fungsi Peristiwa Lambda menggunakan aliran DynamoDB di tempat Akun AWS lain, Anda harus mengonfigurasi aliran menggunakan kebijakan berbasis sumber daya untuk memberikan izin Fungsi Peristiwa Lambda Anda untuk membaca catatan. Untuk mempelajari cara mengonfigurasi streaming untuk akses lintas akun, lihat Berbagi akses dengan fungsi Lambda lintas akun di Panduan Pengembang Amazon DynamoDB.

Setelah mengonfigurasi streaming Anda dengan kebijakan berbasis sumber daya yang memberikan izin yang diperlukan kepada Fungsi Acara Lambda, buat pemetaan sumber peristiwa dengan ARN aliran lintas akun Anda. Anda dapat menemukan ARN streaming di bawah tab Ekspor dan aliran tabel di konsol DynamoDB lintas akun.

Saat menggunakan konsol Lambda, rekatkan ARN aliran langsung ke bidang input tabel DynamoDB di halaman pembuatan pemetaan sumber peristiwa.

Catatan: Cross-region pemicu tidak didukung.

Mengkonfigurasi respons batch sebagian dengan DynamoDB dan Lambda

Sebelum Anda mengonfigurasi respons batch sebagian, Anda harus terlebih dahulu membuat pemetaan sumber peristiwa DynamoDB. Untuk petunjuk, lihat the section called “Buat pemetaan”.

Ketika menggunakan dan memproses data streaming dari sumber peristiwa, secara default Lambda memeriksa urutan nomor tertinggi batch hanya ketika batch berhasil diselesaikan. Lambda memperlakukan semua hasil lain sebagai sepenuhnya gagal dan mencoba lagi pemrosesan batch hingga batas coba lagi. Untuk memungkinkan keberhasilan parsial saat memproses batch dari aliran, aktifkan `ReportBatchItemFailures`. Mengizinkan keberhasilan sebagian dapat membantu mengurangi jumlah percobaan ulang pada catatan, meskipun itu tidak sepenuhnya mencegah kemungkinan percobaan ulang dalam catatan yang berhasil.

Untuk mengaktifkan `ReportBatchItemFailures`, masukkan nilai enum **`ReportBatchItemFailures`** dalam daftar FunctionResponseTypes. Daftar ini menunjukkan jenis respons mana yang diaktifkan untuk Fungsi Acara Anda. Anda dapat mengonfigurasi daftar ini saat membuat atau memperbarui pemetaan sumber peristiwa.

> **Note**
>
> Bahkan ketika kode fungsi Anda mengembalikan respons kegagalan sebagian batch, respons ini tidak diproses oleh Lambda kecuali `ReportBatchItemFailures` fitur tersebut diaktifkan secara eksplisit untuk pemetaan sumber acara Anda.

Sintaks laporan

Saat mengonfigurasi pelaporan pada kegagalan item batch, kelas `StreamsEventResponse` dikembalikan dengan daftar kegagalan item batch. Anda dapat menggunakan objek

`StreamsEventResponse` untuk mengembalikan nomor urutan catatan gagal pertama dalam batch. Anda juga dapat membuat kelas kustom Anda sendiri menggunakan sintaks respons yang benar. Struktur JSON berikut menunjukkan sintaks respons yang diperlukan:

```
{
  "batchItemFailures": [
        {
            "itemIdentifier": "<SequenceNumber>"
        }
    ]
}
```

> **Note**
>
> Jika `batchItemFailures` array berisi beberapa item, Lambda menggunakan catatan dengan nomor urut terendah sebagai pos pemeriksaan. Lambda kemudian mencoba kembali semua catatan mulai dari pos pemeriksaan itu.

Status berhasil dan gagal

Lambda memperlakukan batch sebagai sepenuhnya berhasil jika Anda mengembalikan salah satu dari berikut:

- Daftar `batchItemFailure` kosong
- Daftar `batchItemFailure` nol
- `EventResponse` kosong
- `EventResponse` nol

Lambda memperlakukan batch sebagai sepenuhnya gagal jika Anda mengembalikan salah satu dari berikut:

- String `itemIdentifier` kosong
- `itemIdentifier` nol
- `itemIdentifier` dengan nama kunci yang buruk

Lambda mencoba kembali kegagalan berdasarkan strategi coba lagi Anda.

Membagi batch

Jika invokasi Anda gagal dan `BisectBatchOnFunctionError` diaktifkan, batch dibagi terlepas dari pengaturan `ReportBatchItemFailures` Anda.

Ketika respons berhasil batch parsial diterima dan kedua `BisectBatchOnFunctionError` dan `ReportBatchItemFailures` diaktifkan, batch dibagi dua di nomor urutan yang dikembalikan dan Lambda mencoba hanya catatan yang tersisa.

Untuk menyederhanakan implementasi logika respons batch sebagian, pertimbangkan untuk menggunakan utilitas Prosesor Batch dari Powertools untuk AWS Lambda, yang secara otomatis menangani kompleksitas ini untuk Anda.

Berikut adalah beberapa contoh kode fungsi yang mengembalikan daftar ID pesan yang gagal dalam batch:

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch DynamoDB dengan Lambda menggunakan.NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text.Json;
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.DynamoDBEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri

namespace AWSLambda_DDB;
```

```
public class Function
{
    public StreamsEventResponse FunctionHandler(DynamoDBEvent dynamoEvent,
 ILambdaContext context)

    {
        context.Logger.LogInformation($"Beginning to process
 {dynamoEvent.Records.Count} records...");
        List<StreamsEventResponse.BatchItemFailure> batchItemFailures = new
 List<StreamsEventResponse.BatchItemFailure>();
        StreamsEventResponse streamsEventResponse = new StreamsEventResponse();

        foreach (var record in dynamoEvent.Records)
        {
            try
            {
                var sequenceNumber = record.Dynamodb.SequenceNumber;
                context.Logger.LogInformation(sequenceNumber);
            }
            catch (Exception ex)
            {
                context.Logger.LogError(ex.Message);
                batchItemFailures.Add(new StreamsEventResponse.BatchItemFailure()
 { ItemIdentifier = record.Dynamodb.SequenceNumber });
            }
        }

        if (batchItemFailures.Count > 0)
        {
            streamsEventResponse.BatchItemFailures = batchItemFailures;
        }

        context.Logger.LogInformation("Stream processing complete.");
        return streamsEventResponse;
    }
}
```

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch DynamoDB dengan Lambda menggunakan Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

type BatchItemFailure struct {
 ItemIdentifier string `json:"ItemIdentifier"`
}

type BatchResult struct {
 BatchItemFailures []BatchItemFailure `json:"BatchItemFailures"`
}

func HandleRequest(ctx context.Context, event events.DynamoDBEvent)
 (*BatchResult, error) {
 var batchItemFailures []BatchItemFailure
 curRecordSequenceNumber := ""

 for _, record := range event.Records {
  // Process your record
  curRecordSequenceNumber = record.Change.SequenceNumber
 }

 if curRecordSequenceNumber != "" {
  batchItemFailures = append(batchItemFailures, BatchItemFailure{ItemIdentifier:
 curRecordSequenceNumber})
```

```
 }

 batchResult := BatchResult{
  BatchItemFailures: batchItemFailures,
 }

 return &batchResult, nil
}

func main() {
 lambda.Start(HandleRequest)
}
```

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch DynamoDB dengan Lambda menggunakan Java.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.DynamodbEvent;
import com.amazonaws.services.lambda.runtime.events.StreamsEventResponse;
import com.amazonaws.services.lambda.runtime.events.models.dynamodb.StreamRecord;

import java.util.ArrayList;
import java.util.List;

public class ProcessDynamodbRecords implements RequestHandler<DynamodbEvent,
 StreamsEventResponse> {

    @Override
```

```
    public StreamsEventResponse handleRequest(DynamodbEvent input, Context
 context) {

        List<StreamsEventResponse.BatchItemFailure> batchItemFailures = new
 ArrayList<>();
        String curRecordSequenceNumber = "";

        for (DynamodbEvent.DynamodbStreamRecord dynamodbStreamRecord :
 input.getRecords()) {
          try {
                //Process your record
                StreamRecord dynamodbRecord = dynamodbStreamRecord.getDynamodb();
                curRecordSequenceNumber = dynamodbRecord.getSequenceNumber();

            } catch (Exception e) {
                /* Since we are working with streams, we can return the failed
 item immediately.
                   Lambda will immediately begin to retry processing from this
 failed item onwards. */
                batchItemFailures.add(new
 StreamsEventResponse.BatchItemFailure(curRecordSequenceNumber));
                return new StreamsEventResponse(batchItemFailures);
            }
        }

       return new StreamsEventResponse();
    }
}
```

JavaScript

SDK untuk JavaScript (v3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch DynamoDB dengan penggunaan Lambda. JavaScript

```
export const handler = async (event) => {
  const records = event.Records;
  let curRecordSequenceNumber = "";

  for (const record of records) {
    try {
      // Process your record
      curRecordSequenceNumber = record.dynamodb.SequenceNumber;
    } catch (e) {
      // Return failed record's sequence number
      return { batchItemFailures: [{ itemIdentifier:
 curRecordSequenceNumber }] };
    }
  }

  return { batchItemFailures: [] };
};
```

Melaporkan kegagalan item batch DynamoDB dengan penggunaan Lambda. TypeScript

```
import {
  DynamoDBBatchResponse,
  DynamoDBBatchItemFailure,
  DynamoDBStreamEvent,
} from "aws-lambda";

export const handler = async (
  event: DynamoDBStreamEvent
): Promise<DynamoDBBatchResponse> => {
  const batchItemFailures: DynamoDBBatchItemFailure[] = [];
  let curRecordSequenceNumber;

  for (const record of event.Records) {
    curRecordSequenceNumber = record.dynamodb?.SequenceNumber;

    if (curRecordSequenceNumber) {
      batchItemFailures.push({
        itemIdentifier: curRecordSequenceNumber,
      });
    }
  }
```

```
  return { batchItemFailures: batchItemFailures };
};
```

PHP

SDK untuk PHP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch DynamoDB dengan Lambda menggunakan PHP.

```
<?php

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\DynamoDb\DynamoDbEvent;
use Bref\Event\Handler as StdHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler implements StdHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handle(mixed $event, Context $context): array
    {
        $dynamoDbEvent = new DynamoDbEvent($event);
```

```
        $this->logger->info("Processing records");

        $records = $dynamoDbEvent->getRecords();
        $failedRecords = [];
        foreach ($records as $record) {
            try {
                $data = $record->getData();
                $this->logger->info(json_encode($data));
                // TODO: Do interesting work based on the new data
            } catch (Exception $e) {
                $this->logger->error($e->getMessage());
                // failed processing the record
                $failedRecords[] = $record->getSequenceNumber();
            }
        }
        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords records");

        // change format for the response
        $failures = array_map(
            fn(string $sequenceNumber) => ['itemIdentifier' => $sequenceNumber],
            $failedRecords
        );

        return [
            'batchItemFailures' => $failures
        ];
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch DynamoDB dengan Lambda menggunakan Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def handler(event, context):
    records = event.get("Records")
    curRecordSequenceNumber = ""

    for record in records:
        try:
            # Process your record
            curRecordSequenceNumber = record["dynamodb"]["SequenceNumber"]
        except Exception as e:
            # Return failed record's sequence number
            return {"batchItemFailures":[{"itemIdentifier":
 curRecordSequenceNumber}]}

    return {"batchItemFailures":[]}
```

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch DynamoDB dengan Lambda menggunakan Ruby.

```
def lambda_handler(event:, context:)
    records = event["Records"]
    cur_record_sequence_number = ""

    records.each do |record|
      begin
        # Process your record
        cur_record_sequence_number = record["dynamodb"]["SequenceNumber"]
      rescue StandardError => e
        # Return failed record's sequence number
        return {"batchItemFailures" => [{"itemIdentifier" =>
 cur_record_sequence_number}]}
      end
    end

    {"batchItemFailures" => []}
  end
```

Rust

SDK for Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch DynamoDB dengan Lambda menggunakan Rust.

```
use aws_lambda_events::{
    event::dynamodb::{Event, EventRecord, StreamRecord},
    streams::{DynamoDbBatchItemFailure, DynamoDbEventResponse},
};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

/// Process the stream record
fn process_record(record: &EventRecord) -> Result<(), Error> {
    let stream_record: &StreamRecord = &record.change;
```

```
    // process your stream record here...
    tracing::info!("Data: {:?}", stream_record);

    Ok(())
}

/// Main Lambda handler here...
async fn function_handler(event: LambdaEvent<Event>) ->
 Result<DynamoDbEventResponse, Error> {
    let mut response = DynamoDbEventResponse {
        batch_item_failures: vec![],
    };

    let records = &event.payload.records;

    if records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(response);
    }

    for record in records {
        tracing::info!("EventId: {}", record.event_id);

        // Couldn't find a sequence number
        if record.change.sequence_number.is_none() {
            response.batch_item_failures.push(DynamoDbBatchItemFailure {
                item_identifier: Some("".to_string()),
            });
            return Ok(response);
        }

        // Process your record here...
        if process_record(record).is_err() {
            response.batch_item_failures.push(DynamoDbBatchItemFailure {
                item_identifier: record.change.sequence_number.clone(),
            });
            /* Since we are working with streams, we can return the failed item
 immediately.
            Lambda will immediately begin to retry processing from this failed
 item onwards. */
            return Ok(response);
        }
    }
```

```
    tracing::info!("Successfully processed {} record(s)", records.len());

    Ok(response)
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        // disable printing the name of the module in every log line.
        .with_target(false)
        // disabling time is handy because CloudWatch will add the ingestion
 time.
        .without_time()
        .init();

    run(service_fn(function_handler)).await
}
```

Menggunakan Powertools untuk AWS Lambda prosesor batch

Utilitas prosesor batch dari Powertools untuk AWS Lambda secara otomatis menangani logika respons batch paral, mengurangi kompleksitas penerapan pelaporan kegagalan batch. Berikut adalah contoh menggunakan prosesor batch:

Python

> **Note**
>
> Untuk contoh lengkap dan petunjuk penyiapan, lihat dokumentasi prosesor batch.

Memproses rekaman AWS Lambda aliran DynamoDB dengan prosesor batch.

```
import json
from aws_lambda_powertools import Logger
from aws_lambda_powertools.utilities.batch import BatchProcessor, EventType,
 process_partial_response
from aws_lambda_powertools.utilities.data_classes import DynamoDBStreamEvent
```

```
from aws_lambda_powertools.utilities.typing import LambdaContext

processor = BatchProcessor(event_type=EventType.DynamoDBStreams)
logger = Logger()

def record_handler(record):
    logger.info(record)
    # Your business logic here
    # Raise an exception to mark this record as failed

def lambda_handler(event, context: LambdaContext):
    return process_partial_response(
        event=event,
        record_handler=record_handler,
        processor=processor,
        context=context
    )
```

TypeScript

> **Note**
>
> Untuk contoh lengkap dan petunjuk penyiapan, lihat dokumentasi prosesor batch.

Memproses rekaman AWS Lambda aliran DynamoDB dengan prosesor batch.

```
import { BatchProcessor, EventType, processPartialResponse } from '@aws-lambda-
powertools/batch';
import { Logger } from '@aws-lambda-powertools/logger';
import type { DynamoDBStreamEvent, Context } from 'aws-lambda';

const processor = new BatchProcessor(EventType.DynamoDBStreams);
const logger = new Logger();

const recordHandler = async (record: any): Promise<void> => {
    logger.info('Processing record', { record });
    // Your business logic here
    // Throw an error to mark this record as failed
};

export const handler = async (event: DynamoDBStreamEvent, context: Context) => {
    return processPartialResponse(event, recordHandler, processor, {
```

```
        context,
    });
};
```

Java

> **Note**
>
> Untuk contoh lengkap dan petunjuk penyiapan, lihat dokumentasi prosesor batch.

Memproses rekaman AWS Lambda aliran DynamoDB dengan prosesor batch.

```
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.DynamodbEvent;
import com.amazonaws.services.lambda.runtime.events.StreamsEventResponse;
import software.amazon.lambda.powertools.batch.BatchMessageHandlerBuilder;
import software.amazon.lambda.powertools.batch.handler.BatchMessageHandler;

public class DynamoDBStreamBatchHandler implements RequestHandler<DynamodbEvent,
 StreamsEventResponse> {

    private final BatchMessageHandler<DynamodbEvent, StreamsEventResponse> handler;

    public DynamoDBStreamBatchHandler() {
        handler = new BatchMessageHandlerBuilder()
                .withDynamoDbBatchHandler()
                .buildWithRawMessageHandler(this::processMessage);
    }

    @Override
    public StreamsEventResponse handleRequest(DynamodbEvent ddbEvent, Context
 context) {
        return handler.processBatch(ddbEvent, context);
    }

    private void processMessage(DynamodbEvent.DynamodbStreamRecord
 dynamodbStreamRecord, Context context) {
        // Process the change record
    }
}
```

.NET

> **Note**
>
> Untuk contoh lengkap dan petunjuk penyiapan, lihat dokumentasi prosesor batch.

Memproses rekaman AWS Lambda aliran DynamoDB dengan prosesor batch.

```
using System;
using System.Threading;
using System.Threading.Tasks;
using Amazon.Lambda.Core;
using Amazon.Lambda.DynamoDBEvents;
using Amazon.Lambda.Serialization.SystemTextJson;
using AWS.Lambda.Powertools.BatchProcessing;

[assembly: LambdaSerializer(typeof(DefaultLambdaJsonSerializer))]

namespace HelloWorld;

public class Customer
{
    public string? CustomerId { get; set; }
    public string? Name { get; set; }
    public string? Email { get; set; }
    public DateTime CreatedAt { get; set; }
}

internal class TypedDynamoDbRecordHandler : ITypedRecordHandler<Customer>
{
    public async Task<RecordHandlerResult> HandleAsync(Customer customer,
 CancellationToken cancellationToken)
    {
        if (string.IsNullOrEmpty(customer.Email))
        {
            throw new ArgumentException("Customer email is required");
        }

        return await Task.FromResult(RecordHandlerResult.None);
    }
}
```

```
public class Function
{
    [BatchProcessor(TypedRecordHandler = typeof(TypedDynamoDbRecordHandler))]
    public BatchItemFailuresResponse HandlerUsingTypedAttribute(DynamoDBEvent _)
    {
        return TypedDynamoDbStreamBatchProcessor.Result.BatchItemFailuresResponse;
    }
}
```

Menyimpan catatan yang dibuang untuk sumber peristiwa DynamoDB di Lambda

Penanganan kesalahan untuk pemetaan sumber peristiwa DynamoDB tergantung pada apakah kesalahan terjadi sebelum Fungsi Peristiwa dipanggil atau selama pemanggilan fungsi:

- Sebelum pemanggilan: Jika pemetaan sumber peristiwa Lambda tidak dapat menjalankan fungsi karena pembatasan atau masalah lain, fungsi tersebut akan mencoba lagi hingga catatan kedaluwarsa atau melebihi usia maksimum yang dikonfigurasi pada pemetaan sumber peristiwa (). MaximumRecordAgeInSeconds
- Selama pemanggilan: Jika fungsi dipanggil tetapi menampilkan kesalahan, Lambda mencoba ulang hingga catatan kedaluwarsa, melebihi usia maksimum (MaximumRecordAgeInSeconds), atau mencapai kuota coba ulang yang dikonfigurasi (). MaximumRetryAttempts Untuk kesalahan fungsi, Anda juga dapat mengonfigurasi BisectBatchOnFunctionError, yang membagi batch yang gagal menjadi dua batch yang lebih kecil, mengisolasi catatan buruk dan menghindari batas waktu. Memisahkan batch tidak menghabiskan kuota coba lagi.

Jika tindakan penanganan kesalahan gagal, Lambda membuang rekaman dan melanjutkan pemrosesan batch dari aliran. Dengan pengaturan default, ini berarti rekaman yang buruk dapat memblokir pemrosesan pada shard yang terpengaruh hingga selama satu hari. Untuk menghindari hal ini, konfigurasikan pemetaan sumber kejadian fungsi Anda dengan jumlah percobaan ulang yang wajar dan usia maksimum rekaman yang sesuai dengan kasus penggunaan Anda.

Mengonfigurasi tujuan untuk pemanggilan yang gagal

Untuk menyimpan catatan pemanggilan pemetaan sumber peristiwa yang gagal, tambahkan tujuan ke pemetaan sumber peristiwa Fungsi Acara Anda. Setiap catatan yang dikirim ke tujuan adalah dokumen JSON yang berisi metadata tentang pemanggilan yang gagal. Untuk tujuan Amazon S3, Lambda juga mengirimkan seluruh catatan pemanggilan bersama dengan metadata. Anda dapat

mengonfigurasi topik Amazon SNS, antrian Amazon SQS, bucket Amazon S3, atau Kafka sebagai tujuan.

Dengan tujuan Amazon S3, Anda dapat menggunakan fitur Pemberitahuan Acara Amazon S3 untuk menerima notifikasi saat objek diunggah ke bucket S3 tujuan Anda. Anda juga dapat mengonfigurasi Pemberitahuan Acara S3 untuk memanggil Fungsi Peristiwa Lambda lain untuk melakukan pemrosesan otomatis pada batch yang gagal.

Peran eksekusi Anda harus memiliki izin untuk tujuan:

- Untuk tujuan SQS: sqs: SendMessage
- Untuk tujuan SNS: SNS: Publish
- Untuk tujuan S3: s3: PutObject dan s3: ListBucket
- Untuk tujuan Kafka: kafka-cluster: WriteData

Anda dapat mengonfigurasi topik Kafka sebagai tujuan gagal untuk pemetaan sumber acara Kafka Anda. Ketika Lambda tidak dapat memproses catatan setelah percobaan ulang yang melelahkan atau ketika catatan melebihi usia maksimum, Lambda mengirimkan catatan yang gagal ke topik Kafka yang ditentukan untuk diproses nanti. Lihat the section called “Tujuan kegagalan Kafka”.

Jika Anda telah mengaktifkan enkripsi dengan kunci KMS Anda sendiri untuk tujuan S3, peran eksekusi fungsi Anda juga harus memiliki izin untuk memanggil kms:. GenerateDataKey Jika kunci KMS dan tujuan bucket S3 berada di akun yang berbeda dari Fungsi Peristiwa Lambda dan peran eksekusi, konfigurasikan kunci KMS untuk mempercayai peran eksekusi agar memungkinkan kms:. GenerateDataKey

Untuk mengonfigurasi tujuan saat gagal menggunakan konsol, ikuti langkah-langkah berikut:

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih fungsi.
3. Di bagian Gambaran umum fungsi, pilih Tambahkan tujuan.
4. Untuk Sumber, pilih Pemanggilan pemetaan sumber acara.
5. Untuk pemetaan sumber peristiwa, pilih sumber peristiwa yang dikonfigurasi untuk fungsi ini.
6. Untuk Kondisi, pilih On failure. Untuk pemanggilan pemetaan sumber peristiwa, ini adalah satu-satunya kondisi yang diterima.
7. Untuk tipe Tujuan, pilih jenis tujuan yang Lambda kirimkan catatan pemanggilan.
8. Untuk Tujuan, pilih sumber daya.

9. Pilih Simpan.

Anda juga dapat mengonfigurasi tujuan pada kegagalan menggunakan AWS Command Line Interface (AWS CLI). Misalnya, perintah create-event-source-mapping berikut menambahkan pemetaan sumber peristiwa dengan tujuan kegagalan SQS ke: `MyFunction`

```
aws lambda create-event-source-mapping \
--function-name "MyFunction" \
--event-source-arn arn:aws:dynamodb:us-east-2:123456789012:table/my-table/
stream/2024-06-10T19:26:16.525 \
--destination-config '{"OnFailure": {"Destination": "arn:aws:sqs:us-
east-1:123456789012:dest-queue"}}'
```

Perintah update-event-source-mapping berikut memperbarui pemetaan sumber peristiwa untuk mengirim catatan pemanggilan yang gagal ke tujuan SNS setelah dua percobaan ulang, atau jika catatan berusia lebih dari satu jam.

```
aws lambda update-event-source-mapping \
--uuid f89f8514-cdd9-4602-9e1f-01a5b77d449b \
--maximum-retry-attempts 2 \
--maximum-record-age-in-seconds 3600 \
--destination-config '{"OnFailure": {"Destination": "arn:aws:sns:us-
east-1:123456789012:dest-topic"}}'
```

Pengaturan yang diperbarui diterapkan secara asinkron dan tidak muncul dalam output hingga proses selesai. Gunakan perintah get-event-source-mapping untuk melihat status saat ini.

Untuk menghapus tujuan, berikan string kosong sebagai argumen ke `destination-config` parameter:

```
aws lambda update-event-source-mapping \
--uuid f89f8514-cdd9-4602-9e1f-01a5b77d449b \
--destination-config '{"OnFailure": {"Destination": ""}}'
```

Praktik terbaik keamanan untuk tujuan Amazon S3

Menghapus bucket S3 yang dikonfigurasi sebagai tujuan tanpa menghapus tujuan dari konfigurasi fungsi Anda dapat menimbulkan risiko keamanan. Jika pengguna lain mengetahui nama bucket tujuan Anda, mereka dapat membuat ulang bucket di bucket tersebut. Akun AWS Catatan

pemanggilan yang gagal akan dikirim ke bucket mereka, yang berpotensi mengekspos data dari fungsi Anda.

> **Warning**
>
> Untuk memastikan bahwa catatan pemanggilan dari Fungsi Peristiwa Anda tidak dapat dikirim ke bucket S3 di bucket lain Akun AWS, tambahkan kondisi ke peran eksekusi fungsi Anda yang membatasi `s3:PutObject` izin ke bucket di akun Anda.

Contoh berikut menunjukkan kebijakan IAM yang membatasi `s3:PutObject` izin fungsi Anda ke bucket di akun Anda. Kebijakan ini juga memberi Lambda `s3:ListBucket` izin yang dibutuhkan untuk menggunakan bucket S3 sebagai tujuan.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "S3BucketResourceAccountWrite",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::*/*",
                "arn:aws:s3:::*"
            ],
            "Condition": {
                "StringEquals": {
                    "s3:ResourceAccount": "111122223333"
                }
            }
        }
    ]
}
```

Untuk menambahkan kebijakan izin ke peran eksekusi fungsi Anda menggunakan Konsol Manajemen AWS atau AWS CLI, lihat instruksi dalam prosedur berikut:

Console

Untuk menambahkan kebijakan izin ke peran eksekusi fungsi (konsol)

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih Lambda Event Function yang peran eksekusinya ingin Anda ubah.
3. Di tab Konfigurasi, pilih Izin.
4. Di tab Peran eksekusi, pilih nama Peran fungsi Anda untuk membuka halaman konsol IAM peran.
5. Tambahkan kebijakan izin ke peran dengan melakukan hal berikut:
   a. Di panel Kebijakan izin, pilih Tambahkan izin, lalu pilih Buat kebijakan sebaris.
   b. Di Editor kebijakan, pilih JSON.
   c. Rekatkan kebijakan yang ingin Anda tambahkan ke editor (ganti JSON yang ada), lalu pilih Berikutnya.
   d. Di bawah Detail kebijakan, masukkan nama Kebijakan.
   e. Pilih Buat kebijakan.

AWS CLI

Untuk menambahkan kebijakan izin ke peran eksekusi Fungsi Peristiwa (CLI)

1. Buat dokumen kebijakan JSON dengan izin yang diperlukan dan simpan di direktori lokal.
2. Gunakan perintah IAM `put-role-policy` CLI untuk menambahkan izin ke peran eksekusi fungsi Anda. Jalankan perintah berikut dari direktori tempat Anda menyimpan dokumen kebijakan JSON dan ganti nama peran, nama kebijakan, dan dokumen kebijakan dengan nilai Anda sendiri.

```
aws iam put-role-policy \
--role-name my_lambda_role \
--policy-name LambdaS3DestinationPolicy \
--policy-document file://my_policy.json
```

Contoh catatan pemanggilan Amazon SNS dan Amazon SQS

Contoh berikut menunjukkan catatan pemanggilan yang dikirim Lambda ke tujuan SQS atau SNS untuk aliran DynamoDB.

```
{
    "requestContext": {
        "requestId": "316aa6d0-8154-xmpl-9af7-85d5f4a6bc81",
        "functionArn": "arn:aws:lambda:us-east-2:123456789012:function:myfunction",
        "condition": "RetryAttemptsExhausted",
        "approximateInvokeCount": 1
    },
    "responseContext": {
        "statusCode": 200,
        "executedVersion": "$LATEST",
        "functionError": "Unhandled"
    },
    "version": "1.0",
    "timestamp": "2019-11-14T00:13:49.717Z",
    "DDBStreamBatchInfo": {
        "shardId": "shardId-00000001573689847184-864758bb",
        "startSequenceNumber": "800000000003126276362",
        "endSequenceNumber": "800000000003126276362",
        "approximateArrivalOfFirstRecord": "2019-11-14T00:13:19Z",
        "approximateArrivalOfLastRecord": "2019-11-14T00:13:19Z",
        "batchSize": 1,
        "streamArn": "arn:aws:dynamodb:us-east-2:123456789012:table/mytable/
stream/2019-11-14T00:04:06.388"
    }
}
```

Anda dapat menggunakan informasi ini guna mengambil rekaman yang terpengaruh dari aliran untuk pemecahan masalah. Rekaman aktual tidak disertakan, jadi Anda harus memproses rekaman ini dan mengambilnya dari aliran sebelum kedaluwarsa dan hilang.

Contoh catatan pemanggilan Amazon S3

Contoh berikut menunjukkan rekaman pemanggilan yang dikirim Lambda ke bucket S3 untuk aliran DynamoDB. Selain semua bidang dari contoh sebelumnya untuk tujuan SQS dan SNS, `payload` bidang berisi catatan pemanggilan asli sebagai string JSON yang lolos.

```
{
```

```
    "requestContext": {
        "requestId": "316aa6d0-8154-xmpl-9af7-85d5f4a6bc81",
        "functionArn": "arn:aws:lambda:us-east-2:123456789012:function:myfunction",
        "condition": "RetryAttemptsExhausted",
        "approximateInvokeCount": 1
    },
    "responseContext": {
        "statusCode": 200,
        "executedVersion": "$LATEST",
        "functionError": "Unhandled"
    },
    "version": "1.0",
    "timestamp": "2019-11-14T00:13:49.717Z",
    "DDBStreamBatchInfo": {
        "shardId": "shardId-00000001573689847184-864758bb",
        "startSequenceNumber": "800000000003126276362",
        "endSequenceNumber": "800000000003126276362",
        "approximateArrivalOfFirstRecord": "2019-11-14T00:13:19Z",
        "approximateArrivalOfLastRecord": "2019-11-14T00:13:19Z",
        "batchSize": 1,
        "streamArn": "arn:aws:dynamodb:us-east-2:123456789012:table/mytable/
stream/2019-11-14T00:04:06.388"
    },
    "payload": "<Whole Event>" // Only available in S3
}
```

Objek S3 yang berisi catatan pemanggilan menggunakan konvensi penamaan berikut:

```
aws/lambda/<ESM-UUID>/<shardID>/YYYY/MM/DD/YYYY-MM-DDTHH.MM.SS-<Random UUID>
```

Menerapkan pemrosesan aliran DynamoDB stateful di Lambda

Lambda Event Functions dapat menjalankan aplikasi pemrosesan aliran kontinu. Aliran merupakan data tidak terbatas yang mengalir terus-menerus melalui aplikasi Anda. Untuk menganalisis informasi dari input yang terus diperbarui ini, Anda dapat mengikat catatan yang disertakan menggunakan jendela yang didefinisikan dalam hal waktu.

Jatuh jendela adalah jendela waktu yang berbeda yang membuka dan menutup secara berkala. Secara default, pemanggilan Lambda tidak memiliki status — Anda tidak dapat menggunakannya untuk memproses data di beberapa pemanggilan berkelanjutan tanpa database eksternal. Namun, dengan jendela yang jatuh, Anda dapat mempertahankan status Anda di seluruh pemanggilan.

Status ini berisi hasil agregat pesan yang sebelumnya diproses untuk jendela saat ini. Status Anda bisa maksimal 1 MB untuk setiap pecahan. Jika melebihi ukuran tersebut, Lambda mengakhiri jendela lebih awal.

Setiap rekaman dalam aliran milik jendela tertentu. Lambda memproses setiap rekaman setidaknya sekali, tetapi tidak menjamin bahwa setiap catatan diproses hanya sekali. Dalam kasus yang jarang terjadi, seperti penanganan kesalahan, beberapa catatan mungkin diproses lebih dari sekali. Catatan selalu diproses secara berurutan pertama kali. Jika catatan diproses lebih dari satu kali, mereka mungkin diproses rusak.

Agregasi dan pemrosesan

Fungsi yang dikelola pengguna Anda dipanggil baik untuk agregasi maupun untuk memproses hasil akhir agregasi tersebut. Lambda mengumpulkan semua catatan yang diterima di jendela. Anda dapat menerima catatan ini dalam beberapa batch, masing-masing sebagai invokasi terpisah. Setiap invokasi menerima status.

Jadi, saat menggunakan jendela tumbling, respons Fungsi Peristiwa Lambda Anda harus berisi `state` properti. Jika respons tidak berisi `state` properti, Lambda menganggap ini sebagai pemanggilan yang gagal. Untuk memenuhi kondisi ini, fungsi Anda dapat mengembalikan `TimeWindowEventResponse` objek, yang memiliki bentuk JSON berikut:

Example Nilai **`TimeWindowEventResponse`**

```
{
    "state": {
        "1": 282,
        "2": 715
    },
    "batchItemFailures": []
}
```

> **Note**
>
> Untuk fungsi Java, sebaiknya gunakan a `Map<String, String>` untuk mewakili status.

Di akhir jendela, bendera `isFinalInvokeForWindow` diatur `true` untuk menunjukkan bahwa ini adalah status akhir dan siap untuk diproses. Setelah pemrosesan, jendela selesai, dan invokasi akhir Anda selesai, lalu status dihapus.

Di akhir jendela Anda, Lambda menggunakan pemrosesan akhir untuk tindakan pada hasil agregasi. Pemrosesan akhir Anda dipanggil secara sinkron. Setelah pemanggilan berhasil, fungsi Anda memeriksa nomor urut dan pemrosesan aliran berlanjut. Jika pemanggilan tidak berhasil, Fungsi Acara Lambda Anda menangguhkan pemrosesan lebih lanjut hingga pemanggilan berhasil.

Example DynamodbTimeWindowEvent

```
{
   "Records":[
      {
         "eventID":"1",
         "eventName":"INSERT",
         "eventVersion":"1.0",
         "eventSource":"aws:dynamodb",
         "awsRegion":"us-east-1",
         "dynamodb":{
            "Keys":{
               "Id":{
                  "N":"101"
               }
            },
            "NewImage":{
               "Message":{
                  "S":"New item!"
               },
               "Id":{
                  "N":"101"
               }
            },
            "SequenceNumber":"111",
            "SizeBytes":26,
            "StreamViewType":"NEW_AND_OLD_IMAGES"
         },
         "eventSourceARN":"stream-ARN"
      },
      {
         "eventID":"2",
         "eventName":"MODIFY",
         "eventVersion":"1.0",
         "eventSource":"aws:dynamodb",
         "awsRegion":"us-east-1",
         "dynamodb":{
```

```
            "Keys":{
               "Id":{
                  "N":"101"
               }
            },
            "NewImage":{
               "Message":{
                  "S":"This item has changed"
               },
               "Id":{
                  "N":"101"
               }
            },
            "OldImage":{
               "Message":{
                  "S":"New item!"
               },
               "Id":{
                  "N":"101"
               }
            },
            "SequenceNumber":"222",
            "SizeBytes":59,
            "StreamViewType":"NEW_AND_OLD_IMAGES"
         },
         "eventSourceARN":"stream-ARN"
      },
      {
         "eventID":"3",
         "eventName":"REMOVE",
         "eventVersion":"1.0",
         "eventSource":"aws:dynamodb",
         "awsRegion":"us-east-1",
         "dynamodb":{
            "Keys":{
               "Id":{
                  "N":"101"
               }
            },
            "OldImage":{
               "Message":{
                  "S":"This item has changed"
               },
               "Id":{
```

```
                    "N":"101"
                 }
              },
              "SequenceNumber":"333",
              "SizeBytes":38,
              "StreamViewType":"NEW_AND_OLD_IMAGES"
           },
           "eventSourceARN":"stream-ARN"
        }
    ],
     "window": {
         "start": "2020-07-30T17:00:00Z",
         "end": "2020-07-30T17:05:00Z"
     },
     "state": {
         "1": "state1"
     },
     "shardId": "shard123456789",
     "eventSourceARN": "stream-ARN",
     "isFinalInvokeForWindow": false,
     "isWindowTerminatedEarly": false
}
```

Konfigurasi

Anda dapat mengonfigurasi jendela berguling saat membuat atau memperbarui pemetaan sumber peristiwa. Untuk mengkonfigurasi jendela tumbling, tentukan jendela dalam hitungan detik (TumblingWindowInSeconds). Contoh berikut AWS Command Line Interface (AWS CLI) perintah membuat pemetaan sumber acara streaming yang memiliki jendela jatuh 120 detik. Fungsi Peristiwa Lambda yang didefinisikan untuk agregasi dan pemrosesan diberi nama. `tumbling-window-example-function`

```
aws lambda create-event-source-mapping \
--event-source-arn arn:aws:dynamodb:us-east-2:123456789012:table/my-table/
stream/2024-06-10T19:26:16.525 \
--function-name tumbling-window-example-function \
--starting-position TRIM_HORIZON \
--tumbling-window-in-seconds 120
```

Lambda menentukan jatuh batas jendela berguling berdasarkan waktu ketika catatan dimasukkan ke dalam aliran. Semua catatan memiliki stempel waktu perkiraan yang tersedia yang digunakan Lambda dalam penentuan batas.

Agregasi jendela berguling tidak mendukung shard ulang. Ketika shard berakhir, Lambda menganggap jendela ditutup, dan shard anak memulai jendela mereka sendiri dalam status baru.

Jendela berguling sepenuhnya mendukung kebijakan coba lagi yang ada `maxRetryAttempts` dan `maxRecordAge`.

Example Handler.py — Agregasi dan pemrosesan

Fungsi Python berikut menunjukkan cara untuk menggabungkan, lalu memproses status akhir Anda:

```
def lambda_handler(event, context):
    print('Incoming event: ', event)
    print('Incoming state: ', event['state'])

#Check if this is the end of the window to either aggregate or process.
    if event['isFinalInvokeForWindow']:
        # logic to handle final state of the window
        print('Destination invoke')
    else:
        print('Aggregate invoke')

#Check for early terminations
    if event['isWindowTerminatedEarly']:
        print('Window terminated early')

    #Aggregation logic
    state = event['state']
    for record in event['Records']:
        state[record['dynamodb']['NewImage']['Id']] = state.get(record['dynamodb']
['NewImage']['Id'], 0) + 1

    print('Returning state: ', state)
    return {'state': state}
```

Parameter Lambda untuk pemetaan sumber peristiwa Amazon DynamoDB

Semua jenis sumber peristiwa Lambda berbagi operasi yang sama CreateEventSourceMappingdan UpdateEventSourceMappingAPI. Namun, hanya beberapa parameter yang berlaku untuk DynamoDB Streams.

Parameter	Diperlukan	Default	Catatan
BatchSize	T	100	Maksimum: 10.000.
BisectBatchOnFunctionError	T	false	none
DestinationConfig	T	N/A	Antrian Amazon SQS standar atau tujuan topik Amazon SNS standar untuk catatan yang dibuang
Diaktifkan	T	true	none
EventSourceArn	Y	N/A	ARN dari aliran data atau konsumen aliran
FilterCriteria	T	N/A	Kontrol peristiwa mana yang dikirim Lambda ke Fungsi Acara Anda
FunctionName	Y	N/A	none
FunctionResponseTypes	T	N/A	Agar fungsi Anda melaporkan kegagalan tertentu dalam satu batch, sertakan nilainya `ReportBatchItemFailures` `FunctionResponseTypes` . Untuk informasi selengkapnya, lihat Mengkonfigurasi respons batch

Parameter	Diperlukan	Default	Catatan
			sebagian dengan DynamoDB dan Lambda.
MaximumBatchingWindowInSeconds	T	0	none
MaximumRecordAgeInSeconds	T	-1	-1 berarti tak terbatas: catatan gagal dicoba ulang sampai catatan kedaluwarsa. Batas retensi data untuk DynamoDB Streams adalah 24 jam. Minimal: -1 Maksimal: 604.800
MaximumRetryAttempts	T	-1	-1 berarti tak terbatas: catatan gagal dicoba ulang sampai catatan kedaluwarsa Minimal: 0 Maksimum: 10.000.
ParallelizationFactor	T	1	Maksimal: 10
StartingPosition	Y	N/A	TRIM_HORIZON
TumblingWindowInSeconds	T	N/A	Minimal: 0 Maksimal: 900

Menggunakan pemfilteran acara dengan sumber acara DynamoDB

Anda dapat menggunakan pemfilteran peristiwa untuk mengontrol rekaman mana dari aliran atau antrian yang dikirim Lambda ke Fungsi Acara Anda. Untuk informasi umum tentang cara kerja penyaringan acara, lihatthe section called “Pemfilteran peristiwa”.

Bagian ini berfokus pada penyaringan acara untuk sumber acara DynamoDB.

> **Note**
>
> Pemetaan sumber peristiwa DynamoDB hanya mendukung pemfilteran pada kunci. `dynamodb`

Topik

- Acara DynamoDB
- Pemfilteran dengan atribut tabel
- Memfilter dengan ekspresi Boolean
- Menggunakan operator Exists
- Format JSON untuk pemfilteran DynamoDB

Acara DynamoDB

Misalkan Anda memiliki tabel DynamoDB dengan `CustomerName` kunci utama dan atribut dan. `AccountManager` `PaymentTerms` Berikut ini menunjukkan contoh catatan dari aliran tabel DynamoDB Anda.

```
{
      "eventID": "1",
      "eventVersion": "1.0",
      "dynamodb": {
          "ApproximateCreationDateTime": "1678831218.0",
          "Keys": {
              "CustomerName": {
                  "S": "AnyCompany Industries"
              }
          },
          "NewImage": {
              "AccountManager": {
```

```
                "S": "Pat Candella"
            },
            "PaymentTerms": {
                "S": "60 days"
            },
            "CustomerName": {
                "S": "AnyCompany Industries"
            }
        },
        "SequenceNumber": "111",
        "SizeBytes": 26,
        "StreamViewType": "NEW_IMAGE"
    }
}
```

Untuk memfilter berdasarkan nilai kunci dan atribut dalam tabel DynamoDB Anda, gunakan kunci `dynamodb` dalam catatan. Bagian berikut memberikan contoh untuk berbagai jenis filter.

Pemfilteran dengan tombol tabel

Misalkan Anda ingin Fungsi Acara Anda memproses hanya catatan-catatan di mana kunci utamanya `CustomerName` adalah "AnyCompany Industri.” `FilterCriteria`Objeknya adalah sebagai berikut.

```
{
    "Filters": [
        {
            "Pattern": "{ \"dynamodb\" : { \"Keys\" : { \"CustomerName\" : { \"S\" :
 [ \"AnyCompany Industries\" ] } } } }"
        }
    ]
}
```

Untuk kejelasan tambahan, berikut adalah nilai filter yang `Pattern` diperluas di JSON biasa.

```
{
    "dynamodb": {
        "Keys": {
            "CustomerName": {
                "S": [ "AnyCompany Industries" ]
                }
            }
        }
```

```
}
```

Anda dapat menambahkan filter menggunakan konsol, AWS CLI atau AWS SAM templat.

Console

Untuk menambahkan filter ini menggunakan konsol, ikuti instruksi Melampirkan kriteria filter ke pemetaan sumber peristiwa (konsol) dan masukkan string berikut untuk kriteria Filter.

```
{ "dynamodb" : { "Keys" : { "CustomerName" : { "S" : [ "AnyCompany
 Industries" ] } } } }
```

AWS CLI

Untuk membuat pemetaan sumber peristiwa baru dengan kriteria filter ini menggunakan AWS Command Line Interface (AWS CLI), jalankan perintah berikut.

```
aws lambda create-event-source-mapping \
    --function-name my-function \
    --event-source-arn arn:aws:dynamodb:us-east-2:123456789012:table/my-table \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"dynamodb\" : { \"Keys\" :
 { \"CustomerName\" : { \"S\" : [ \"AnyCompany Industries\" ] } } } }"}]}'
```

Untuk menambahkan kriteria filter ini ke pemetaan sumber peristiwa yang ada, jalankan perintah berikut.

```
aws lambda update-event-source-mapping \
    --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"dynamodb\" : { \"Keys\" :
 { \"CustomerName\" : { \"S\" : [ \"AnyCompany Industries\" ] } } } }"}]}'
```

AWS SAM

Untuk menambahkan filter ini menggunakan AWS SAM, tambahkan cuplikan berikut ke template YAMB untuk sumber acara Anda.

```
FilterCriteria:
   Filters:
     - Pattern: '{ "dynamodb" : { "Keys" : { "CustomerName" : { "S" : [ "AnyCompany
 Industries" ] } } } }'
```

Pemfilteran dengan atribut tabel

Dengan DynamoDB, Anda juga dapat menggunakan `NewImage` tombol `OldImage` and untuk memfilter nilai atribut. Misalkan Anda ingin memfilter catatan di mana `AccountManager` atribut dalam gambar tabel terbaru adalah “Pat Candella” atau “Shirley Rodriguez.” `FilterCriteria`Objeknya adalah sebagai berikut.

```
{
    "Filters": [
        {
            "Pattern": "{ \"dynamodb\" : { \"NewImage\" : { \"AccountManager\" : { \"S
\" : [ \"Pat Candella\", \"Shirley Rodriguez\" ] } } } }"
        }
    ]
}
```

Untuk kejelasan tambahan, berikut adalah nilai filter yang `Pattern` diperluas di JSON biasa.

```
{
    "dynamodb": {
        "NewImage": {
            "AccountManager": {
                "S": [ "Pat Candella", "Shirley Rodriguez" ]
            }
        }
    }
}
```

Anda dapat menambahkan filter menggunakan konsol, AWS CLI atau AWS SAM templat.

Console

Untuk menambahkan filter ini menggunakan konsol, ikuti instruksi Melampirkan kriteria filter ke pemetaan sumber peristiwa (konsol) dan masukkan string berikut untuk kriteria Filter.

```
{ "dynamodb" : { "NewImage" : { "AccountManager" : { "S" : [ "Pat Candella",
 "Shirley Rodriguez" ] } } } }
```

AWS CLI

Untuk membuat pemetaan sumber peristiwa baru dengan kriteria filter ini menggunakan AWS Command Line Interface (AWS CLI), jalankan perintah berikut.

```
aws lambda create-event-source-mapping \
    --function-name my-function \
    --event-source-arn arn:aws:dynamodb:us-east-2:123456789012:table/my-table \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"dynamodb\" : { \"NewImage
\" : { \"AccountManager\" : { \"S\" : [ \"Pat Candella\", \"Shirley Rodriguez
\" ] } } } }"}]}'
```

Untuk menambahkan kriteria filter ini ke pemetaan sumber peristiwa yang ada, jalankan perintah berikut.

```
aws lambda update-event-source-mapping \
    --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"dynamodb\" : { \"NewImage
\" : { \"AccountManager\" : { \"S\" : [ \"Pat Candella\", \"Shirley Rodriguez
\" ] } } } }"}]}'
```

AWS SAM

Untuk menambahkan filter ini menggunakan AWS SAM, tambahkan cuplikan berikut ke template YAMB untuk sumber acara Anda.

```
FilterCriteria:
  Filters:
    - Pattern: '{ "dynamodb" : { "NewImage" : { "AccountManager" : { "S" : [ "Pat
 Candella", "Shirley Rodriguez" ] } } } }'
```

Memfilter dengan ekspresi Boolean

Anda juga dapat membuat filter menggunakan ekspresi Boolean AND. Ekspresi ini dapat mencakup parameter kunci dan atribut tabel Anda. Misalkan Anda ingin memfilter catatan di mana `NewImage` nilainya `AccountManager` adalah “Pat Candella” dan `OldImage` nilainya adalah “Terry Whitlock”. `FilterCriteria`Objeknya adalah sebagai berikut.

```
{
    "Filters": [
        {
            "Pattern": "{ \"dynamodb\" : { \"NewImage\" : { \"AccountManager\" : { \"S
\" : [ \"Pat Candella\" ] } } } , \"dynamodb\" : { \"OldImage\" : { \"AccountManager
\" : { \"S\" : [ \"Terry Whitlock\" ] } } } }"
```

```
        }
    ]
}
```

Untuk kejelasan tambahan, berikut adalah nilai filter yang `Pattern` diperluas di JSON biasa.

```
{
    "dynamodb" : {
        "NewImage" : {
            "AccountManager" : {
                "S" : [
                    "Pat Candella"
                ]
            }
        }
    },
    "dynamodb": {
        "OldImage": {
            "AccountManager": {
                "S": [
                    "Terry Whitlock"
                ]
            }
        }
    }
}
```

Anda dapat menambahkan filter menggunakan konsol, AWS CLI atau AWS SAM templat.

Console

Untuk menambahkan filter ini menggunakan konsol, ikuti instruksi Melampirkan kriteria filter ke pemetaan sumber peristiwa (konsol) dan masukkan string berikut untuk kriteria Filter.

```
{ "dynamodb" : { "NewImage" : { "AccountManager" : { "S" : [ "Pat
 Candella" ] } } } , "dynamodb" : { "OldImage" : { "AccountManager" : { "S" :
 [ "Terry Whitlock" ] } } } }
```

AWS CLI

Untuk membuat pemetaan sumber peristiwa baru dengan kriteria filter ini menggunakan AWS Command Line Interface (AWS CLI), jalankan perintah berikut.

```
aws lambda create-event-source-mapping \
    --function-name my-function \
    --event-source-arn arn:aws:dynamodb:us-east-2:123456789012:table/my-table \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"dynamodb\" : { \"NewImage
\" : { \"AccountManager\" : { \"S\" : [ \"Pat Candella\" ] } } } , \"dynamodb\" :
 { \"OldImage\" : { \"AccountManager\" : { \"S\" : [ \"Terry Whitlock\" ] } } } }
 "}]}'
```

Untuk menambahkan kriteria filter ini ke pemetaan sumber peristiwa yang ada, jalankan perintah berikut.

```
aws lambda update-event-source-mapping \
    --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"dynamodb\" : { \"NewImage
\" : { \"AccountManager\" : { \"S\" : [ \"Pat Candella\" ] } } } , \"dynamodb\" :
 { \"OldImage\" : { \"AccountManager\" : { \"S\" : [ \"Terry Whitlock\" ] } } } }
 "}]}'
```

AWS SAM

Untuk menambahkan filter ini menggunakan AWS SAM, tambahkan cuplikan berikut ke template YAMB untuk sumber acara Anda.

```
FilterCriteria:
  Filters:
    - Pattern: '{ "dynamodb" : { "NewImage" : { "AccountManager" : { "S" : [ "Pat
 Candella" ] } } } , "dynamodb" : { "OldImage" : { "AccountManager" : { "S" :
 [ "Terry Whitlock" ] } } } }'
```

> **Note**
>
> DynamoDB event filtering tidak mendukung penggunaan operator numerik (sama numerik dan rentang numerik). Bahkan jika item dalam tabel Anda disimpan sebagai angka, parameter ini dikonversi ke string di objek catatan JSON.

Menggunakan operator Exists

Karena cara objek peristiwa JSON dari DynamoDB terstruktur, menggunakan operator Exists memerlukan perhatian khusus. Operator Exists hanya bekerja pada node daun di acara JSON, jadi

jika pola filter Anda menggunakan Exists untuk menguji node perantara, itu tidak akan berfungsi. Pertimbangkan item tabel DynamoDB berikut:

```
{
  "UserID": {"S": "12345"},
  "Name": {"S": "John Doe"},
  "Organizations": {"L": [
      {"S":"Sales"},
      {"S":"Marketing"},
      {"S":"Support"}
    ]
  }
}
```

Anda mungkin ingin membuat pola filter seperti berikut ini yang akan menguji peristiwa yang berisi`"Organizations"`:

```
{ "dynamodb" : { "NewImage" : { "Organizations" : [ { "exists": true } ] } } }
```

Namun, pola filter ini tidak akan pernah mengembalikan kecocokan karena `"Organizations"` bukan simpul daun. Contoh berikut menunjukkan bagaimana benar menggunakan operator Exists untuk membangun pola filter yang diinginkan:

```
{ "dynamodb" : { "NewImage" : {"Organizations": {"L": {"S": [ {"exists":
 true } ] } } } } }
```

Format JSON untuk pemfilteran DynamoDB

Untuk memfilter peristiwa dengan benar dari sumber DynamoDB, bidang data dan kriteria filter Anda untuk bidang data `dynamodb` () harus dalam format JSON yang valid. Jika salah satu bidang tidak dalam format JSON yang valid, Lambda akan menghapus pesan atau melempar pengecualian. Tabel berikut merangkum perilaku spesifik:

Format data masuk	Format pola filter untuk properti data	Tindakan yang dihasilkan
JSON yang valid	JSON yang valid	Filter Lambda berdasarkan kriteria filter Anda.

Format data masuk	Format pola filter untuk properti data	Tindakan yang dihasilkan
JSON yang valid	Tidak ada pola filter untuk properti data	Filter Lambda (hanya pada properti metadata lainnya) berdasarkan kriteria filter Anda.
JSON yang valid	Non-JSON	Lambda melempar pengecualian pada saat pembuatan atau pembaruan pemetaan sumber acara. Pola filter untuk properti data harus dalam format JSON yang valid.
Non-JSON	JSON yang valid	Lambda menjatuhkan rekor.
Non-JSON	Tidak ada pola filter untuk properti data	Filter Lambda (hanya pada properti metadata lainnya) berdasarkan kriteria filter Anda.
Non-JSON	Non-JSON	Lambda melempar pengecualian pada saat pembuatan atau pembaruan pemetaan sumber acara. Pola filter untuk properti data harus dalam format JSON yang valid.

Tutorial: Menggunakan AWS Lambda dengan aliran Amazon DynamoDB

Dalam tutorial ini, Anda membuat Lambda Event Function untuk mengkonsumsi peristiwa dari aliran Amazon DynamoDB.

Prasyarat

Instal AWS Command Line Interface

Jika Anda belum menginstal AWS Command Line Interface, ikuti langkah-langkah di Menginstal atau memperbarui versi terbaru AWS CLI untuk menginstalnya.

Tutorial ini membutuhkan terminal baris perintah atau shell untuk menjalankan perintah. Di Linux dan macOS, gunakan shell dan manajer paket pilihan Anda.

> **Note**
>
> Di Windows, beberapa perintah Bash CLI yang biasa Anda gunakan dengan Lambda (`zip`seperti) tidak didukung oleh terminal bawaan sistem operasi. Untuk mendapatkan Windows-integrated versi Ubuntu dan Bash, instal Subsistem Windows untuk Linux.

Buat peran eksekusi

Buat peran eksekusi yang memberikan izin fungsi Anda untuk mengakses AWS sumber daya.

Untuk membuat peran eksekusi

1. Buka halaman peran di konsol IAM.
2. Pilih Buat peran.
3. Buat peran dengan properti berikut.
   - Entitas tepercaya – Lambda.
   - Izin – AWSLambdaDynamoDBExecutionRole.
   - Nama peran – **`lambda-dynamodb-role`**.

Ini AWSLambdaDynamoDBExecutionRolememiliki izin bahwa fungsi perlu membaca item dari DynamoDB dan menulis log ke Log. CloudWatch

Buat Fungsi Acara

Buat Fungsi Peristiwa Lambda yang memproses peristiwa DynamoDB Anda. Kode fungsi menulis beberapa data peristiwa yang masuk ke CloudWatch Log.

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara DynamoDB dengan Lambda menggunakan.NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text.Json;
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.DynamoDBEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri

namespace AWSLambda_DDB;

public class Function
{
    public void FunctionHandler(DynamoDBEvent dynamoEvent, ILambdaContext
 context)
    {
        context.Logger.LogInformation($"Beginning to process
 {dynamoEvent.Records.Count} records...");

        foreach (var record in dynamoEvent.Records)
        {
            context.Logger.LogInformation($"Event ID: {record.EventID}");
            context.Logger.LogInformation($"Event Name: {record.EventName}");

            context.Logger.LogInformation(JsonSerializer.Serialize(record));
        }

        context.Logger.LogInformation("Stream processing complete.");
```

```
    }
}
```

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara DynamoDB dengan Lambda menggunakan Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "github.com/aws/aws-lambda-go/lambda"
 "github.com/aws/aws-lambda-go/events"
 "fmt"
)

func HandleRequest(ctx context.Context, event events.DynamoDBEvent) (*string,
 error) {
 if len(event.Records) == 0 {
  return nil, fmt.Errorf("received empty event")
 }

 for _, record := range event.Records {
   LogDynamoDBRecord(record)
 }

 message := fmt.Sprintf("Records processed: %d", len(event.Records))
 return &message, nil
}

func main() {
```

```
 lambda.Start(HandleRequest)
}

func LogDynamoDBRecord(record events.DynamoDBEventRecord){
 fmt.Println(record.EventID)
 fmt.Println(record.EventName)
 fmt.Printf("%+v\n", record.Change)
}
```

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara DynamoDB dengan Lambda menggunakan Java.

```
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.DynamodbEvent;
import
 com.amazonaws.services.lambda.runtime.events.DynamodbEvent.DynamodbStreamRecord;
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;

public class example implements RequestHandler<DynamodbEvent, Void> {

    private static final Gson GSON = new
 GsonBuilder().setPrettyPrinting().create();

    @Override
    public Void handleRequest(DynamodbEvent event, Context context) {
        System.out.println(GSON.toJson(event));
        event.getRecords().forEach(this::logDynamoDBRecord);
        return null;
    }

    private void logDynamoDBRecord(DynamodbStreamRecord record) {
```

```
        System.out.println(record.getEventID());
        System.out.println(record.getEventName());
        System.out.println("DynamoDB Record: " +
 GSON.toJson(record.getDynamodb()));
    }
}
```

JavaScript

SDK untuk JavaScript (v3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara DynamoDB dengan Lambda menggunakan. JavaScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
    console.log(JSON.stringify(event, null, 2));
    event.Records.forEach(record => {
        logDynamoDBRecord(record);
    });
};

const logDynamoDBRecord = (record) => {
    console.log(record.eventID);
    console.log(record.eventName);
    console.log(`DynamoDB Record: ${JSON.stringify(record.dynamodb)}`);
};
```

Mengkonsumsi acara DynamoDB dengan Lambda menggunakan. TypeScript

```
export const handler = async (event, context) => {
    console.log(JSON.stringify(event, null, 2));
    event.Records.forEach(record => {
        logDynamoDBRecord(record);
```

```
    });
}
const logDynamoDBRecord = (record) => {
    console.log(record.eventID);
    console.log(record.eventName);
    console.log(`DynamoDB Record: ${JSON.stringify(record.dynamodb)}`);
};
```

PHP

SDK untuk PHP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara DynamoDB dengan Lambda menggunakan PHP.

```
<?php

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\DynamoDb\DynamoDbEvent;
use Bref\Event\DynamoDb\DynamoDbHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler extends DynamoDbHandler
{
    private StderrLogger $logger;

    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
```

```
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handleDynamoDb(DynamoDbEvent $event, Context $context): void
    {
        $this->logger->info("Processing DynamoDb table items");
        $records = $event->getRecords();

        foreach ($records as $record) {
            $eventName = $record->getEventName();
            $keys = $record->getKeys();
            $old = $record->getOldImage();
            $new = $record->getNewImage();

            $this->logger->info("Event Name:".$eventName."\n");
            $this->logger->info("Keys:". json_encode($keys)."\n");
            $this->logger->info("Old Image:". json_encode($old)."\n");
            $this->logger->info("New Image:". json_encode($new));

            // TODO: Do interesting work based on the new data

            // Any exception thrown will be logged and the invocation will be
 marked as failed
        }

        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords items");
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara DynamoDB dengan Lambda menggunakan Python.

```
import json

def lambda_handler(event, context):
    print(json.dumps(event, indent=2))

    for record in event['Records']:
        log_dynamodb_record(record)

def log_dynamodb_record(record):
    print(record['eventID'])
    print(record['eventName'])
    print(f"DynamoDB Record: {json.dumps(record['dynamodb'])}")
```

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara DynamoDB dengan Lambda menggunakan Ruby.

```
def lambda_handler(event:, context:)
    return 'received empty event' if event['Records'].empty?

    event['Records'].each do |record|
      log_dynamodb_record(record)
    end

    "Records processed: #{event['Records'].length}"
  end

  def log_dynamodb_record(record)
```

```
    puts record['eventID']
    puts record['eventName']
    puts "DynamoDB Record: #{JSON.generate(record['dynamodb'])}"
  end
```

Rust

SDK for Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara DynamoDB dengan Lambda menggunakan Rust.

```
use lambda_runtime::{service_fn, tracing, Error, LambdaEvent};
use aws_lambda_events::{
    event::dynamodb::{Event, EventRecord},
   };


// Built with the following dependencies:
//lambda_runtime = "0.11.1"
//serde_json = "1.0"
//tokio = { version = "1", features = ["macros"] }
//tracing = { version = "0.1", features = ["log"] }
//tracing-subscriber = { version = "0.3", default-features = false, features =
 ["fmt"] }
//aws_lambda_events = "0.15.0"

async fn function_handler(event: LambdaEvent<Event>) ->Result<(), Error> {

    let records = &event.payload.records;
    tracing::info!("event payload: {:?}",records);
    if records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(());
    }
```

```
    for record in records{
        log_dynamo_dbrecord(record);
    }

    tracing::info!("Dynamo db records processed");

    // Prepare the response
    Ok(())

}

fn log_dynamo_dbrecord(record: &EventRecord)-> Result<(), Error>{
    tracing::info!("EventId: {}", record.event_id);
    tracing::info!("EventName: {}", record.event_name);
    tracing::info!("DynamoDB Record: {:?}", record.change );
    Ok(())

}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
    .with_max_level(tracing::Level::INFO)
    .with_target(false)
    .without_time()
    .init();

    let func = service_fn(function_handler);
    lambda_runtime::run(func).await?;
    Ok(())

}
```

Untuk membuat Fungsi Acara

1. Salin kode sampel ke file dengan nama `example.js`.
2. Buat paket deployment.

```
zip function.zip example.js
```

3. Buat Fungsi Acara Lambda dengan perintah. `create-function`

```
aws lambda create-function --function-name ProcessDynamoDBRecords \
    --zip-file fileb://function.zip --handler example.handler --runtime nodejs24.x
 \
    --role arn:aws:iam::111122223333:role/lambda-dynamodb-role
```

Uji Fungsi Peristiwa Lambda

Pada langkah ini, Anda memanggil Lambda Event Function Anda secara manual menggunakan perintah `invoke` AWS Lambda CLI dan contoh peristiwa DynamoDB berikut. Salin berikut ini ke dalam file bernama`input.txt`.

Example input.txt

```
{
   "Records":[
      {
         "eventID":"1",
         "eventName":"INSERT",
         "eventVersion":"1.0",
         "eventSource":"aws:dynamodb",
         "awsRegion":"us-east-1",
         "dynamodb":{
            "Keys":{
               "Id":{
                  "N":"101"
               }
            },
            "NewImage":{
               "Message":{
                  "S":"New item!"
               },
               "Id":{
                  "N":"101"
               }
            },
            "SequenceNumber":"111",
            "SizeBytes":26,
            "StreamViewType":"NEW_AND_OLD_IMAGES"
         },
         "eventSourceARN":"stream-ARN"
```

```
    },
    {
       "eventID":"2",
       "eventName":"MODIFY",
       "eventVersion":"1.0",
       "eventSource":"aws:dynamodb",
       "awsRegion":"us-east-1",
       "dynamodb":{
          "Keys":{
             "Id":{
                "N":"101"
             }
          },
          "NewImage":{
             "Message":{
                "S":"This item has changed"
             },
             "Id":{
                "N":"101"
             }
          },
          "OldImage":{
             "Message":{
                "S":"New item!"
             },
             "Id":{
                "N":"101"
             }
          },
          "SequenceNumber":"222",
          "SizeBytes":59,
          "StreamViewType":"NEW_AND_OLD_IMAGES"
       },
       "eventSourceARN":"stream-ARN"
    },
    {
       "eventID":"3",
       "eventName":"REMOVE",
       "eventVersion":"1.0",
       "eventSource":"aws:dynamodb",
       "awsRegion":"us-east-1",
       "dynamodb":{
          "Keys":{
             "Id":{
```

```
                    "N":"101"
                }
            },
            "OldImage":{
                "Message":{
                    "S":"This item has changed"
                },
                "Id":{
                    "N":"101"
                }
            },
            "SequenceNumber":"333",
            "SizeBytes":38,
            "StreamViewType":"NEW_AND_OLD_IMAGES"
         },
         "eventSourceARN":"stream-ARN"
      }
   ]
}
```

Jalankan perintah `invoke` berikut.

```
aws lambda invoke --function-name ProcessDynamoDBRecords \
    --cli-binary-format raw-in-base64-out \
    --payload file://input.txt outputfile.txt
```

cli-binary-formatOpsi ini diperlukan jika Anda menggunakan AWS CLI versi 2. Untuk menjadikan ini pengaturan default, jalankan`aws configure set cli-binary-format raw-in-base64-out`. Untuk informasi selengkapnya, lihat opsi baris perintah global yang AWS CLI didukung di Panduan AWS Command Line Interface Pengguna untuk Versi 2.

Fungsi mengembalikan `message` string dalam badan respons.

Verifikasikan output dalam `outputfile.txt` file.

Buat tabel DynamoDB dengan aliran yang diaktifkan

Buat tabel Amazon DynamoDB dengan aliran yang diaktifkan.

Untuk membuat tabel DynamoDB

1. Buka Konsol DynamoDB.

2. Pilih Buat tabel.
3. Buat tabel dengan pengaturan berikut.
   - Nama tabel – **`lambda-dynamodb-stream`**
   - Kunci utama – **`id`** (string)
4. Pilih Buat.

Untuk mengaktifkan stream

1. Buka Konsol DynamoDB.
2. Pilih Tables.
3. Pilih tabel lambda-dynamodb-stream.
4. Di bawah Ekspor dan aliran, pilih detail aliran DynamoDB.
5. Pilih Nyalakan.
6. Untuk tipe Tampilan, pilih Atribut kunci saja.
7. Pilih Aktifkan aliran.

Tulis ARN stream. Anda memerlukan ini di langkah berikutnya ketika Anda mengaitkan aliran dengan Fungsi Acara Lambda Anda. Untuk informasi selengkapnya tentang mengaktifkan aliran, lihat Menangkap aktivitas tabel dengan DynamoDB Streams.

Tambahkan sumber acara di AWS Lambda

Buat pemetaan sumber peristiwa di AWS Lambda. Pemetaan sumber peristiwa ini mengaitkan aliran DynamoDB dengan Fungsi Peristiwa Lambda Anda. Setelah Anda membuat pemetaan sumber acara ini, AWS Lambda mulai polling aliran.

Jalankan perintah AWS CLI `create-event-source-mapping` berikut. Setelah perintah dijalankan, catat UUID. Anda akan memerlukan UUID ini untuk merujuk ke pemetaan sumber kejadian dalam perintah apa pun, misalnya, saat menghapus pemetaan sumber kejadian.

```
aws lambda create-event-source-mapping --function-name ProcessDynamoDBRecords \
    --batch-size 100 --starting-position LATEST --event-source DynamoDB-stream-arn
```

Ini menciptakan pemetaan antara aliran DynamoDB yang ditentukan dan Fungsi Peristiwa Lambda. Anda dapat mengaitkan aliran DynamoDB dengan beberapa Fungsi Peristiwa Lambda, dan

mengaitkan Fungsi Peristiwa Lambda yang sama dengan beberapa aliran. Namun, Fungsi Acara Lambda membagikan throughput baca untuk streaming yang mereka bagikan.

Anda bisa mendapatkan daftar pemetaan sumber kejadian dengan menjalankan perintah berikut.

```
aws lambda list-event-source-mappings
```

Daftar tersebut mengembalikan semua pemetaan sumber kejadian yang Anda buat, dan antara lain menampilkan `LastProcessingResult` untuk setiap pemetaan. Bidang ini digunakan untuk memberikan pesan informatif jika terjadi masalah. Nilai seperti `No records processed` (menunjukkan bahwa AWS Lambda belum memulai polling atau bahwa tidak ada catatan dalam aliran) dan `OK` (menunjukkan AWS Lambda berhasil membaca catatan dari aliran dan memanggil Fungsi Peristiwa Lambda Anda) menunjukkan bahwa tidak ada masalah. Jika ada masalah, Anda akan menerima pesan kesalahan.

Jika Anda memiliki banyak pemetaan sumber kejadian, gunakan parameter nama fungsi untuk mempersempit hasil.

```
aws lambda list-event-source-mappings --function-name ProcessDynamoDBRecords
```

Uji penyiapan

Uji pengalaman end-to-end. Saat Anda melakukan pembaruan tabel, DynamoDB menuliskan catatan peristiwa ke aliran. Saat melakukan AWS Lambda polling streaming, ia mendeteksi catatan baru dalam aliran dan memanggil Fungsi Peristiwa Lambda Anda atas nama Anda dengan meneruskan peristiwa ke fungsi.

1. Di konsol DynamoDB, tambahkan, perbarui, dan hapus item di tabel. DynamoDB menuliskan catatan tindakan ini ke aliran.
2. AWS Lambda polling aliran dan ketika mendeteksi pembaruan ke aliran, ia memanggil Fungsi Peristiwa Lambda Anda dengan meneruskan data peristiwa yang ditemukannya di aliran.
3. Fungsi Anda berjalan dan membuat log di Amazon CloudWatch. Anda dapat memverifikasi log yang dilaporkan di CloudWatch konsol Amazon.

Langkah selanjutnya

Tutorial ini menunjukkan dasar-dasar pemrosesan peristiwa aliran DynamoDB dengan Lambda. Untuk beban kerja produksi, pertimbangkan untuk menerapkan logika respons batch sebagian untuk

menangani kegagalan rekaman individu dengan lebih efisien. Utilitas prosesor batch dari Powertools for AWS Lambda tersedia dalam Python TypeScript,, .NET, dan Java dan memberikan solusi yang kuat untuk ini, secara otomatis menangani kompleksitas respons batch paral dan mengurangi jumlah percobaan ulang untuk catatan yang berhasil diproses.

Bersihkan sumber daya Anda

Sekarang Anda dapat menghapus sumber daya yang Anda buat untuk tutorial ini, kecuali Anda ingin mempertahankannya. Dengan menghapus AWS sumber daya yang tidak lagi Anda gunakan, Anda mencegah tagihan yang tidak perlu ke Anda Akun AWS.

Untuk menghapus Fungsi Acara Lambda

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih Fungsi Acara yang Anda buat.
3. Pilih Tindakan, Hapus.
4. Ketik **`confirm`** kolom input teks dan pilih Hapus.

Untuk menghapus peran eksekusi

1. Buka halaman Peran dari konsol IAM.
2. Pilih peran eksekusi yang Anda buat.
3. Pilih Hapus.
4. Masukkan nama peran di bidang input teks dan pilih Hapus.

Untuk menghapus tabel DynamoDB

1. Buka halaman Tabel di konsol DynamoDB.
2. Pilih tabel yang Anda buat.
3. Pilih Hapus.
4. Masukkan **`delete`** di kotak teks.
5. Pilih Hapus tabel.

Memproses peristiwa siklus hidup Amazon EC2 dengan Fungsi Peristiwa Lambda

Anda dapat menggunakannya AWS Lambda untuk memproses peristiwa siklus hidup dari Amazon Elastic Compute Cloud dan mengelola sumber daya Amazon EC2. Amazon EC2 mengirimkan peristiwa ke Amazon EventBridge (CloudWatch Acara) untuk peristiwa siklus hidup seperti saat instans mengubah status, saat snapshot volume Amazon Elastic Block Store selesai, atau saat Instans Spot dijadwalkan untuk dihentikan. Anda mengonfigurasi EventBridge (CloudWatch Peristiwa) untuk meneruskan peristiwa tersebut ke Fungsi Peristiwa Lambda untuk diproses.

EventBridge (CloudWatch Peristiwa) memanggil Fungsi Peristiwa Lambda Anda secara asinkron dengan dokumen peristiwa dari Amazon EC2.

Example kejadian siklus hidup instans

```
{
    "version": "0",
    "id": "b6ba298a-7732-2226-xmpl-976312c1a050",
    "detail-type": "EC2 Instance State-change Notification",
    "source": "aws.ec2",
    "account": "111122223333",
    "time": "2019-10-02T17:59:30Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:ec2:us-east-1:111122223333:instance/i-0c314xmplcd5b8173"
    ],
    "detail": {
        "instance-id": "i-0c314xmplcd5b8173",
        "state": "running"
    }
}
```

Untuk detail tentang mengonfigurasi acara, lihatMemanggil Fungsi Acara Lambda sesuai jadwal. Untuk fungsi contoh yang memproses notifikasi snapshot Amazon EBS, lihat EventBridge Penjadwal untuk Amazon EBS.

Anda juga dapat menggunakan AWS SDK untuk mengelola instans dan sumber daya lainnya dengan Amazon EC2 API.

Memberikan izin untuk EventBridge (Acara) CloudWatch

Untuk memproses peristiwa siklus hidup dari Amazon EC2 EventBridge , CloudWatch (Acara) memerlukan izin untuk menjalankan fungsi Anda. Izin ini berasal dari kebijakan berbasis sumber daya milik fungsi. Jika Anda menggunakan konsol EventBridge (CloudWatch Peristiwa) untuk mengonfigurasi pemicu peristiwa, konsol akan memperbarui kebijakan berbasis sumber daya atas nama Anda. Jika tidak, tambahkan pernyataan seperti berikut:

Example pernyataan kebijakan berbasis sumber daya untuk pemberitahuan siklus hidup Amazon EC2

```
{
  "Sid": "ec2-events",
  "Effect": "Allow",
  "Principal": {
    "Service": "events.amazonaws.com"
  },
  "Action": "lambda:InvokeFunction",
  "Resource": "arn:aws:lambda:us-east-1:12456789012:function:my-function",
  "Condition": {
    "ArnLike": {
      "AWS:SourceArn": "arn:aws:events:us-east-1:12456789012:rule/*"
    }
  }
}
```

Untuk menambahkan pernyataan, gunakan `add-permission` AWS CLI perintah.

```
aws lambda add-permission --action lambda:InvokeFunction --statement-id ec2-events \
--principal events.amazonaws.com --function-name my-function --source-arn
 'arn:aws:events:us-east-1:12456789012:rule/*'
```

Jika fungsi Anda menggunakan AWS SDK untuk mengelola resource Amazon EC2, tambahkan izin Amazon EC2 ke peran eksekusi fungsi.

Memproses permintaan Application Load Balancer dengan Lambda

Anda dapat menggunakan Lambda Event Function untuk memproses permintaan dari Application Load Balancer. Elastic Load Balancing mendukung Lambda Event Functions sebagai target untuk Application Load Balancer. Gunakan aturan load balancer untuk merutekan permintaan HTTP ke fungsi, berdasarkan nilai jalur atau header. Memproses permintaan dan mengembalikan respon HTTP dari Lambda Event Function Anda.

Elastic Load Balancing memanggil Fungsi Peristiwa Lambda Anda secara sinkron dengan peristiwa yang berisi badan permintaan dan metadata.

Example Kejadian permintaan Application Load Balancer


```
{
    "requestContext": {
        "elb": {
            "targetGroupArn": "arn:aws:elasticloadbalancing:us-
east-1:123456789012:targetgroup/lambda-279XGJDqGZ5rsrHC2Fjr/49e9d65c45c6791a"
        }
    },
    "httpMethod": "GET",
    "path": "/lambda",
    "queryStringParameters": {
        "query": "1234ABCD"
    },
    "headers": {
        "accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/
webp,image/apng,*/*;q=0.8",
        "accept-encoding": "gzip",
        "accept-language": "en-US,en;q=0.9",
        "connection": "keep-alive",
        "host": "lambda-alb-123578498.us-east-1.elb.amazonaws.com",
        "upgrade-insecure-requests": "1",
        "user-agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
 (KHTML, like Gecko) Chrome/71.0.3578.98 Safari/537.36",
        "x-amzn-trace-id": "Root=1-5c536348-3d683b8b04734faae651f476",
        "x-forwarded-for": "72.12.164.125",
        "x-forwarded-port": "80",
        "x-forwarded-proto": "http",
        "x-imforwards": "20"
    },
    "body": "",
```

```
    "isBase64Encoded": False
}
```

Fungsi Anda memproses kejadian dan mengembalikan dokumen respons ke load balancer dalam JSON. Elastic Load Balancing mengonversi dokumen menjadi respons keberhasilan atau kesalahan HTTP dan mengembalikannya ke pengguna.

Example format dokumen respons

```
{
    "statusCode": 200,
    "statusDescription": "200 OK",
    "isBase64Encoded": False,
    "headers": {
        "Content-Type": "text/html"
    },
    "body": "<h1>Hello from Lambda!</h1>"
}
```

Untuk mengonfigurasi Application Load Balancer sebagai pemicu fungsi, berikan izin kepada Elastic Load Balancing untuk menjalankan fungsi, buat grup target yang merutekan permintaan ke fungsi, dan tambahkan aturan ke load balancer yang mengirim permintaan ke grup target.

Gunakan perintah `add-permission` untuk menambahkan pernyataan izin ke kebijakan berbasis sumber daya milik fungsi Anda.

```
aws lambda add-permission --function-name alb-function \
--statement-id load-balancer --action "lambda:InvokeFunction" \
--principal elasticloadbalancing.amazonaws.com
```

Anda akan melihat output berikut:

```
{
    "Statement": "{\"Sid\":\"load-balancer\",\"Effect\":\"Allow\",\"Principal\":
{\"Service\":\"elasticloadbalancing.amazonaws.com\"},\"Action\":\"lambda:InvokeFunction
\",\"Resource\":\"arn:aws:lambda:us-west-2:123456789012:function:alb-function\"}"
}
```

Untuk petunjuk konfigurasi Application Load Balancer dan grup target, lihat Fungsi Lambda sebagai target dalam Panduan Pengguna untuk Application Load Balancer.

Event Handler dari Powertools untuk AWS Lambda

Event handler dari Powertools for AWS Lambda toolkit menyediakan routing, middleware, konfigurasi CORS, pembuatan spesifikasi OpenAPI, validasi permintaan, penanganan kesalahan, dan fitur berguna lainnya saat menulis Lambda Event Functions yang dipanggil oleh Application Load Balancer. Utilitas Event Handler tersedia untuk Python. Untuk informasi selengkapnya, lihat Event Handler REST API di dokumentasi Powertools for AWS Lambda (Python).

Python

```
import requests
from requests import Response

from aws_lambda_powertools import Logger, Tracer
from aws_lambda_powertools.event_handler import ALBResolver
from aws_lambda_powertools.logging import correlation_paths
from aws_lambda_powertools.utilities.typing import LambdaContext

tracer = Tracer()
logger = Logger()
app = ALBResolver()


@app.get("/todos")
@tracer.capture_method
def get_todos():
    todos: Response = requests.get("https://jsonplaceholder.typicode.com/todos")
    todos.raise_for_status()

    # for brevity, we'll limit to the first 10 only
    return {"todos": todos.json()[:10]}


# You can continue to use other utilities just as before
@logger.inject_lambda_context(correlation_id_path=correlation_paths.APPLICATION_LOAD_BALANCER)
@tracer.capture_lambda_handler
def lambda_handler(event: dict, context: LambdaContext) -> dict:
    return app.resolve(event, context)
```

Memanggil Fungsi Acara Lambda sesuai jadwal

Amazon EventBridge Scheduler adalah penjadwal tanpa server yang dapat Anda gunakan untuk membuat, menjalankan, dan mengelola tugas dari satu layanan terpusat dan terkelola. Dengan EventBridge Scheduler, Anda dapat membuat jadwal menggunakan ekspresi cron dan rate untuk pola berulang, atau mengonfigurasi pemanggilan satu kali. Anda dapat mengatur jendela waktu fleksibel untuk pengiriman, menentukan batas coba lagi, dan mengatur waktu retensi maksimum untuk peristiwa yang belum diproses.

Saat Anda mengatur EventBridge Scheduler dengan Lambda EventBridge , Scheduler memanggil Fungsi Acara Lambda Anda secara asinkron. Anda dapat membuat jadwal dari konsol Lambda (menggunakan alur Add trigger), konsol EventBridge Scheduler, atau. AWS CLI CloudFormation Halaman ini menjelaskan cara menggunakan EventBridge Scheduler untuk memanggil Fungsi Acara Lambda pada jadwal.

Siapkan peran eksekusi

Saat Anda membuat jadwal baru, EventBridge Scheduler harus memiliki izin untuk menjalankan operasi API targetnya atas nama Anda. Anda memberikan izin ini ke EventBridge Scheduler menggunakan peran eksekusi. Kebijakan izin yang Anda lampirkan ke peran eksekusi jadwal menentukan izin yang diperlukan. Izin ini bergantung pada API target yang ingin Anda panggil EventBridge Scheduler.

Saat Anda membuat jadwal menggunakan konsol Lambda atau konsol Scheduler, EventBridge EventBridge Scheduler secara otomatis mengatur peran eksekusi berdasarkan target yang Anda pilih. Jika Anda ingin membuat jadwal menggunakan salah satu SDK EventBridge Penjadwal, atau AWS CLI CloudFormation, Anda harus memiliki peran eksekusi yang ada yang memberikan izin yang diperlukan EventBridge Penjadwal untuk memanggil target. Untuk informasi selengkapnya tentang mengatur peran eksekusi secara manual untuk jadwal Anda, lihat Menyiapkan peran eksekusi di Panduan Pengguna EventBridge Penjadwal.

Buat jadwal dari konsol Lambda

Gunakan alur Add trigger di konsol Lambda untuk membuat jadwal EventBridge Scheduler tanpa meninggalkan konsol.

Untuk membuat pemicu Scheduler menggunakan konsol Lambda

1. Buka halaman Fungsi Acara konsol Lambda.

2. Pilih fungsi yang ingin Anda jadwalkan.
3. Di panel Ikhtisar fungsi, pilih Tambah pemicu.
4. Dari dropdown sumber pemicu, pilih Scheduler.
5. Konfigurasikan pengaturan jadwal:

    a. Nama jadwal — Masukkan nama untuk jadwal Anda.

    b. Grup jadwal - Pilih grup jadwal atau gunakan`default`.

    c. Jenis jadwal - Pilih salah satu dari berikut ini:

    - One-time jadwal - Memanggil fungsi sekali pada tanggal dan waktu yang ditentukan.
    - Jadwal berulang - Memanggil fungsi berulang kali menggunakan ekspresi cron atau rate.

    d. (Opsional) Masukkan payload JSON. Jika Anda tidak memasukkan payload, EventBridge Scheduler menggunakan peristiwa kosong untuk menjalankan fungsi.

    e. Konfigurasikan peran eksekusi. Agar EventBridge Scheduler membuat peran eksekusi baru untuk Anda, pilih Buat peran baru untuk jadwal ini. Kemudian, masukkan nama untuk nama Peran. Jika Anda memilih opsi ini, EventBridge Scheduler membuat kebijakan dengan izin yang diperlukan dan melampirkannya ke peran.
6. Pilih Tambahkan.

> **Tip**
>
> Anda juga dapat mencari**`schedule`**,, **`crontime`**, atau **`recurring`** untuk menemukan pemicu ini di alur Add trigger.

Anda juga dapat menggunakan konsol EventBridge Scheduler untuk membuat jadwal. Dengan konsol EventBridge Scheduler, Anda dapat mengonfigurasi opsi tambahan seperti jendela waktu fleksibel dan antrian huruf mati.

Melihat EventBridge pemicu Scheduler

Semua EventBridge jadwal Scheduler yang menargetkan fungsi Anda muncul di bagian Pemicu pada halaman konfigurasi fungsi di bawah label Scheduler. Ini termasuk jadwal yang dibuat dari salah satu sumber berikut:

- Konsol Lambda (Tambahkan alur pemicu)

- Konsol EventBridge Scheduler
- The AWS CLI, SDK, CloudFormation, AWS SAM (menggunakan jenis `ScheduleV2` acara), atau AWS CDK (menggunakan `aws-scheduler` modul)

> **Note**
>
> Jika jadwal menggunakan kunci terkelola pelanggan untuk enkripsi dengan target Universal Target Invocation (UTI), itu mungkin tidak muncul dalam daftar pemicu. Untuk melihat jadwal ini, gunakan konsol EventBridge Scheduler.

Untuk mengonfirmasi bahwa EventBridge Scheduler memanggil fungsi, periksa log Amazon CloudWatch fungsi.

Sumber daya terkait

Untuk informasi selengkapnya tentang EventBridge Scheduler, lihat berikut ini:

- EventBridge Panduan Pengguna Penjadwal
- EventBridge Referensi API Scheduler
- EventBridge Penetapan Harga Penjadwal

Penggunaan AWS Lambda dengan AWS IoT

AWS IoT menyediakan komunikasi yang aman antara perangkat yang terhubung ke internet (seperti sensor) dan Cloud. AWS Ini memungkinkan Anda untuk mengumpulkan, menyimpan, dan menganalisis data telemetri dari beberapa perangkat.

Anda dapat membuat AWS IoT aturan untuk berinteraksi dengan perangkat Anda Layanan AWS. AWS IoT Rules Engine menyediakan SQL-based bahasa untuk memilih data dari muatan pesan dan mengirim data ke layanan lain, seperti Amazon S3, Amazon DynamoDB, dan. AWS Lambda Anda menentukan aturan untuk menjalankan Fungsi Peristiwa Lambda saat Anda ingin memanggil layanan AWS lain atau layanan pihak ketiga.

Saat pesan IoT yang masuk memicu aturan, AWS IoT memanggil Fungsi Peristiwa Lambda Anda secara asinkron dan meneruskan data dari pesan IoT ke fungsi tersebut.

Contoh berikut menunjukkan pembacaan kelembapan dari sensor rumah kaca. Nilai baris dan pos mengidentifikasi lokasi sensor. Contoh kejadian ini didasarkan pada jenis rumah kaca dalam tutorial Aturan AWS IoT.

Example AWS IoT pesan acara

```
{
    "row" : "10",
    "pos" : "23",
    "moisture" : "75"
}
```

Untuk invokasi asinkron, Lambda mengantrekan pesan dan mencoba lagi jika fungsi Anda mengembalikan kesalahan. Konfigurasikan fungsi Anda dengan tujuan untuk menyimpan kejadian yang tidak dapat diproses oleh fungsi Anda.

Anda harus memberikan izin untuk AWS IoT layanan untuk menjalankan Fungsi Acara Lambda Anda. Gunakan perintah `add-permission` untuk menambahkan pernyataan izin ke kebijakan berbasis sumber daya milik fungsi Anda.

> **Important**
>
> Jika Anda menambahkan izin untuk AWS IoT prinsipal tanpa memberikan `source-arn` atau`source-account`, apa pun Akun AWS yang membuat aturan dengan AWS Lambda

tindakan Anda dapat mengaktifkan aturan untuk memanggil Fungsi AWS Lambda Acara Anda dari AWS IoT. Untuk informasi selengkapnya, lihat izin AWS Lambda.

```
aws lambda add-permission --function-name my-function \
--statement-id iot-events --action "lambda:InvokeFunction" --principal
 iot.amazonaws.com \
--source-arn arn:aws:iot:us-east-1:123456789012:rule/my-rule \
--source-account 123456789012
```

Anda akan melihat output berikut:

```
{
    "Statement": "{\"Sid\":\"iot-events\",\"Effect\":\"Allow\",\"Principal\":
{\"Service\":\"iot.amazonaws.com\"},\"Action\":\"lambda:InvokeFunction\",\"Resource
\":\"arn:aws:lambda:us-east-1:123456789012:function:my-function\",\"Condition\":
{\"ArnLike\":{\"AWS:SourceArn\":\"arn:aws:iot:us-east-1:123456789012:rule/my-rule\"},
\"StringEquals\":{\"AWS:SourceAccount\":\"123456789012\"}}}"
}
```

Untuk informasi selengkapnya tentang cara menggunakan Lambda AWS IoT, lihat Membuat AWS Lambda aturan.

Menggunakan Lambda untuk memproses catatan dari Amazon Kinesis Data Streams

Anda dapat menggunakan Fungsi Peristiwa Lambda untuk memproses rekaman dalam aliran data Amazon Kinesis. Anda dapat memetakan Fungsi Peristiwa Lambda ke konsumen throughput bersama Kinesis Data Streams (iterator standar), atau ke konsumen throughput khusus dengan fan-out yang ditingkatkan. Untuk iterator standar, Lambda melakukan polling rekaman di setiap shard dalam aliran Kinesis Anda dengan menggunakan protokol HTTP. Pemetaan sumber kejadian berbagi throughput baca dengan konsumen lain dari shard tersebut.

Untuk perincian tentang aliran data Kinesis, lihat Data Pembacaan dari Amazon Kinesis Data Streams.

Untuk memulai, lihat the section called “Buat pemetaan”. Anda dapat mengonfigurasi parameter pemetaan sumber peristiwa, menggunakan jendela tumbling untuk agregasi, dan melihat kode fungsi contoh.

> **Note**
>
> Kinesis mengenakan biaya untuk setiap shard dan, untuk keluaran yang ditingkatkan, pembacaan data dari aliran. Untuk perincian harga, lihat harga Amazon Kinesis.

Polling dan batching stream

Lambda membaca rekaman dari aliran data dan memanggil fungsi Anda secara sinkron dengan kejadian yang berisi rekaman aliran. Lambda membaca rekaman dalam batch dan memanggil fungsi Anda untuk memproses rekaman dari batch. Setiap batch berisi catatan dari satu shard/data aliran.

Fungsi Acara Lambda Anda adalah aplikasi konsumen untuk aliran data Anda. Itu memproses satu batch rekaman pada satu waktu dari setiap shard. Anda dapat memetakan Fungsi Peristiwa Lambda ke konsumen throughput bersama (iterator standar), atau ke konsumen throughput khusus dengan fan-out yang ditingkatkan.

- Iterator standar: Lambda polling setiap pecahan dalam aliran Kinesis Anda untuk catatan dengan kecepatan dasar sekali per detik. Saat tersedia lebih banyak rekaman, Lambda melanjutkan pemrosesan batch sampai fungsi dapat menyusul aliran. Pemetaan sumber kejadian berbagi throughput baca dengan konsumen lain dari shard tersebut.

- Peningkatan fan-out: Untuk meminimalkan latensi dan memaksimalkan throughput baca, buat konsumen aliran data dengan fan-out yang disempurnakan. Konsumen dengan keluaran yang ditingkatkan mendapatkan koneksi khusus ke setiap shard yang tidak memengaruhi pembacaan aplikasi lain dari aliran tersebut. Konsumen streaming digunakan HTTP/2 untuk mengurangi latensi dengan mendorong catatan ke Lambda melalui koneksi yang berumur panjang dan dengan mengompresi header permintaan. Anda dapat membuat konsumen aliran dengan API Kinesis RegisterStreamConsumer.

```
aws kinesis register-stream-consumer \
--consumer-name con1 \
--stream-arn arn:aws:kinesis:us-east-2:123456789012:stream/lambda-stream
```

Anda akan melihat output berikut:

```
{
    "Consumer": {
        "ConsumerName": "con1",
        "ConsumerARN": "arn:aws:kinesis:us-east-2:123456789012:stream/lambda-stream/
consumer/con1:1540591608",
        "ConsumerStatus": "CREATING",
        "ConsumerCreationTimestamp": 1540591608.0
    }
}
```

Untuk meningkatkan kecepatan proses fungsi Anda merekam, tambahkan pecahan ke aliran data Anda. Lambda memproses rekaman dalam setiap shard secara berurutan. Lambda menghentikan pemrosesan rekaman tambahan dalam shard jika fungsi Anda mengembalikan kesalahan. Makin banyak shard, makin banyak batch yang diproses sekaligus sehingga mengurangi dampak kesalahan terhadap konkurensi.

Apabila fungsi Anda tidak dapat menskalakan naik untuk menangani jumlah total batch yang bersamaan, mintalah kenaikan kuota atau cadangkan konkurensi untuk fungsi Anda.

Secara default, Lambda memanggil fungsi Anda segera setelah catatan tersedia. Jika batch yang dibaca Lambda dari sumber peristiwa hanya memiliki satu catatan di dalamnya, Lambda hanya mengirimkan satu catatan ke fungsi tersebut. Untuk menghindari menjalankan fungsi dengan sejumlah kecil catatan, Anda dapat memberi tahu sumber acara untuk menyangga catatan hingga 5 menit dengan mengonfigurasi jendela batching. Sebelum menjalankan fungsi, Lambda terus membaca catatan dari sumber acara hingga mengumpulkan batch penuh, jendela batching

kedaluwarsa, atau batch mencapai batas muatan 6 MB. Untuk informasi selengkapnya, lihat Perilaku batching.

> **Warning**
>
> Pemetaan sumber peristiwa Lambda memproses setiap peristiwa setidaknya sekali, dan pemrosesan duplikat catatan dapat terjadi. Untuk menghindari potensi masalah yang terkait dengan duplikat peristiwa, kami sangat menyarankan agar Anda membuat kode fungsi Anda idempoten. Untuk mempelajari lebih lanjut, lihat Bagaimana cara membuat fungsi Lambda saya idempoten di Pusat Pengetahuan. AWS

Lambda tidak menunggu ekstensi yang dikonfigurasi selesai sebelum mengirim batch berikutnya untuk diproses. Dengan kata lain, ekstensi Anda dapat terus berjalan saat Lambda memproses kumpulan catatan berikutnya. Hal ini dapat menyebabkan masalah pembatasan jika Anda melanggar pengaturan atau batasan konkurensi akun Anda. Untuk mendeteksi apakah ini merupakan masalah potensial, pantau fungsi Anda dan periksa apakah Anda melihat metrik konkurensi yang lebih tinggi dari yang diharapkan untuk pemetaan sumber peristiwa Anda. Karena waktu yang singkat di antara pemanggilan, Lambda mungkin secara singkat melaporkan penggunaan konkurensi yang lebih tinggi daripada jumlah pecahan. Ini bisa benar bahkan untuk Lambda Event Functions tanpa ekstensi.

Konfigurasikan ParallelizationFactorpengaturan untuk memproses satu pecahan aliran data Kinesis dengan lebih dari satu pemanggilan Lambda secara bersamaan. Anda dapat menentukan jumlah batch bersamaan yang dipolling Lambda dari pecahan dengan menggunakan faktor paralelisasi dari 1 (default) hingga 10. Misalnya, saat Anda menyetel `ParallelizationFactor` ke 2, Anda dapat memiliki maksimum 200 pemanggilan Lambda bersamaan untuk memproses 100 pecahan data Kinesis (meskipun dalam praktiknya, Anda mungkin melihat nilai yang berbeda untuk metrik). `ConcurrentExecutions` Hal ini membantu meningkatkan skala throughput pemrosesan ketika volume data tidak stabil dan `IteratorAge` tinggi. Saat Anda meningkatkan jumlah batch bersamaan per pecahan, Lambda masih memastikan pemrosesan in-order pada tingkat kunci partisi.

Anda juga dapat menggunakan `ParallelizationFactor` dengan agregasi Kinesis. Perilaku pemetaan sumber acara bergantung pada apakah Anda menggunakan fan-out yang disempurnakan:

- Tanpa fan-out yang ditingkatkan: Semua peristiwa di dalam acara agregat harus memiliki kunci partisi yang sama. Kunci partisi juga harus cocok dengan peristiwa agregat. Jika peristiwa di dalam peristiwa agregat memiliki kunci partisi yang berbeda, Lambda tidak dapat menjamin pemrosesan peristiwa secara berurutan dengan kunci partisi.

- Dengan fan-out yang disempurnakan: Pertama, Lambda menerjemahkan peristiwa agregat ke dalam acara individualnya. Acara agregat dapat memiliki kunci partisi yang berbeda dari peristiwa yang dikandungnya. Namun, peristiwa yang tidak sesuai dengan kunci partisi dijatuhkan dan hilang. Lambda tidak memproses peristiwa ini, dan tidak mengirimnya ke tujuan kegagalan yang dikonfigurasi.

Contoh peristiwa

Example

```
{
    "Records": [
        {
            "kinesis": {
                "kinesisSchemaVersion": "1.0",
                "partitionKey": "1",
                "sequenceNumber":
 "49590338271490256608559692538361571095921575989136588898",
                "data": "SGVsbG8sIHRoaXMgaXMgYSB0ZXN0Lg==",
                "approximateArrivalTimestamp": 1545084650.987
            },
            "eventSource": "aws:kinesis",
            "eventVersion": "1.0",
            "eventID":
 "shardId-000000000006:49590338271490256608559692538361571095921575989136588898",
            "eventName": "aws:kinesis:record",
            "invokeIdentityArn": "arn:aws:iam::123456789012:role/lambda-role",
            "awsRegion": "us-east-2",
            "eventSourceARN": "arn:aws:kinesis:us-east-2:123456789012:stream/lambda-
stream"
        },
        {
            "kinesis": {
                "kinesisSchemaVersion": "1.0",
                "partitionKey": "1",
                "sequenceNumber":
 "49590338271490256608559692540925702759324208523137515618",
                "data": "VGhpcyBpcyBvbmx5IGEgdGVzdC4=",
                "approximateArrivalTimestamp": 1545084711.166
            },
            "eventSource": "aws:kinesis",
            "eventVersion": "1.0",
```

```
            "eventID":
 "shardId-000000000006:49590338271490256608559692540925702759324208523137515618",
            "eventName": "aws:kinesis:record",
            "invokeIdentityArn": "arn:aws:iam::123456789012:role/lambda-role",
            "awsRegion": "us-east-2",
            "eventSourceARN": "arn:aws:kinesis:us-east-2:123456789012:stream/lambda-
stream"
        }
    ]
}
```

Memproses rekaman Amazon Kinesis Data Streams dengan Lambda

Untuk memproses rekaman Amazon Kinesis Data Streams dengan Lambda, buat pemetaan sumber peristiwa Lambda. Anda dapat memetakan Fungsi Peristiwa Lambda ke iterator standar atau konsumen penggemar yang disempurnakan. Untuk informasi selengkapnya, lihat Polling dan batching stream.

Buat pemetaan sumber acara Kinesis

Untuk memanggil Fungsi Peristiwa Lambda Anda dengan catatan dari aliran data Anda, buat pemetaan sumber peristiwa. Anda dapat membuat beberapa pemetaan sumber peristiwa untuk memproses data yang sama dengan beberapa Fungsi Peristiwa Lambda, atau untuk memproses item dari beberapa aliran data dengan satu fungsi. Saat memproses item dari beberapa aliran, setiap batch berisi catatan hanya dari satu pecahan atau aliran.

Anda dapat mengonfigurasi pemetaan sumber peristiwa untuk memproses catatan dari aliran yang berbeda. Akun AWS Untuk mempelajari selengkapnya, lihat the section called “Cross-account pemetaan”.

Sebelum Anda membuat pemetaan sumber peristiwa, Anda perlu memberikan izin Fungsi Peristiwa Lambda Anda untuk membaca dari aliran data Kinesis. Lambda memerlukan izin berikut untuk mengelola sumber daya yang terkait dengan aliran data Kinesis Anda:

- kinesis: DescribeStream
- kinesis: DescribeStreamSummary
- kinesis: GetRecords
- kinesis: GetShardIterator
- kinesis: ListShards
- kinesis: SubscribeToShard

Kebijakan AWS terkelola AWSLambdaKinesisExecutionRolemencakup izin ini. Tambahkan kebijakan terkelola ini ke fungsi Anda seperti yang dijelaskan dalam prosedur berikut.

> **Note**
>
> - Anda tidak memerlukan `kinesis:ListStreams` izin untuk membuat dan mengelola pemetaan sumber acara untuk Kinesis. Namun, jika Anda membuat pemetaan sumber peristiwa di konsol dan Anda tidak memiliki izin ini, Anda tidak akan dapat memilih aliran Kinesis dari daftar tarik-turun dan konsol menampilkan kesalahan. Untuk membuat pemetaan sumber peristiwa, Anda harus memasukkan Nama Sumber Daya Amazon (ARN) secara manual dari aliran Anda.
> - Lambda membuat `kinesis:GetRecords` dan panggilan `kinesis:GetShardIterator` API saat mencoba kembali pemanggilan yang gagal.

Konsol Manajemen AWS

Untuk menambahkan izin Kinesis ke Fungsi Acara

1. Buka halaman Fungsi Acara konsol Lambda dan pilih fungsi Anda.
2. Di tab Konfigurasi, pilih Izin.
3. Di panel Peran eksekusi, di bawah Nama peran, pilih tautan ke peran eksekusi fungsi Anda. Tautan ini membuka halaman untuk peran itu di konsol IAM.
4. Di panel Kebijakan izin, pilih Tambahkan izin, lalu pilih Lampirkan kebijakan.
5. Di bidang pencarian, masukkan**`AWSLambdaKinesisExecutionRole`**.
6. Pilih kotak centang di samping kebijakan dan pilih Tambah izin.

AWS CLI

Untuk menambahkan izin Kinesis ke Fungsi Acara

- Jalankan perintah CLI berikut untuk menambahkan `AWSLambdaKinesisExecutionRole` kebijakan ke peran eksekusi fungsi Anda:

```
aws iam attach-role-policy \
--role-name MyFunctionRole \
```

```
--policy-arn arn:aws:iam::aws:policy/service-role/AWSLambdaKinesisExecutionRole
```

AWS SAM

Untuk menambahkan izin Kinesis ke Fungsi Acara

- Dalam definisi fungsi Anda, tambahkan `Policies` properti seperti yang ditunjukkan pada contoh berikut:

```
Resources:
  MyFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: ./my-function/
      Handler: index.handler
      Runtime: nodejs24.x
      Policies:
        - AWSLambdaKinesisExecutionRole
```

Setelah mengonfigurasi izin yang diperlukan, buat pemetaan sumber acara.

Konsol Manajemen AWS

Untuk membuat pemetaan sumber peristiwa Kinesis

1. Buka halaman Fungsi Acara konsol Lambda dan pilih fungsi Anda.
2. Di panel Ikhtisar fungsi, pilih Tambah pemicu.
3. Di bawah Konfigurasi pemicu, untuk sumbernya, pilih Kinesis.
4. Pilih aliran Kinesis yang ingin Anda buat pemetaan sumber acara dan, secara opsional, konsumen aliran Anda.
5. (Opsional) edit ukuran Batch, Posisi awal, dan jendela Batch untuk pemetaan sumber acara Anda.
6. Pilih Tambahkan.

Saat membuat pemetaan sumber peristiwa dari konsol, peran IAM Anda harus memiliki izin kinesis: ListStreams dan kinesis:. ListStreamConsumers

AWS CLI

Untuk membuat pemetaan sumber peristiwa Kinesis

- Jalankan perintah CLI berikut untuk membuat pemetaan sumber peristiwa Kinesis. Pilih ukuran batch Anda sendiri dan posisi awal sesuai dengan kasus penggunaan Anda.

```
aws lambda create-event-source-mapping \
--function-name MyFunction \
--event-source-arn arn:aws:kinesis:us-east-2:123456789012:stream/lambda-stream \
--starting-position LATEST \
--batch-size 100
```

Untuk menentukan jendela batching, tambahkan `--maximum-batching-window-in-seconds` opsi. Untuk informasi selengkapnya tentang penggunaan parameter ini dan parameter lainnya, lihat create-event-source-mapping di Referensi Perintah.AWS CLI

AWS SAM

Untuk membuat pemetaan sumber peristiwa Kinesis

- Dalam definisi fungsi Anda, tambahkan `KinesisEvent` properti seperti yang ditunjukkan pada contoh berikut:

```
Resources:
  MyFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: ./my-function/
      Handler: index.handler
      Runtime: nodejs24.x
      Policies:
        - AWSLambdaKinesisExecutionRole
      Events:
        KinesisEvent:
          Type: Kinesis
          Properties:
            Stream: !GetAtt MyKinesisStream.Arn
            StartingPosition: LATEST
            BatchSize: 100

  MyKinesisStream:
```

```
    Type: AWS::Kinesis::Stream
    Properties:
      ShardCount: 1
```

Untuk mempelajari selengkapnya tentang membuat pemetaan sumber peristiwa untuk Kinesis Data Streams di AWS SAM, lihat Kinesis dalam Panduan Pengembang.AWS Serverless Application Model

Posisi awal polling dan streaming

Ketahuilah bahwa polling streaming selama pembuatan dan pembaruan pemetaan sumber acara pada akhirnya konsisten.

- Selama pembuatan pemetaan sumber acara, mungkin diperlukan beberapa menit untuk memulai acara polling dari aliran.
- Selama pembaruan pemetaan sumber acara, mungkin diperlukan beberapa menit untuk menghentikan dan memulai kembali acara pemungutan suara dari aliran.

Perilaku ini berarti bahwa jika Anda menentukan `LATEST` sebagai posisi awal untuk aliran, pemetaan sumber peristiwa dapat melewatkan peristiwa selama pembuatan atau pembaruan. Untuk memastikan bahwa tidak ada peristiwa yang terlewatkan, tentukan posisi awal aliran sebagai `TRIM_HORIZON` atau`AT_TIMESTAMP`.

Membuat pemetaan sumber peristiwa lintas akun

Amazon Kinesis Data Streams mendukung kebijakan berbasis sumber daya. Karena itu, Anda dapat memproses data yang dicerna ke dalam aliran dalam satu Akun AWS dengan Fungsi Peristiwa Lambda di akun lain.

Untuk membuat pemetaan sumber peristiwa untuk Fungsi Peristiwa Lambda menggunakan aliran Kinesis yang Akun AWS berbeda, Anda harus mengonfigurasi aliran menggunakan kebijakan berbasis sumber daya untuk memberikan izin Fungsi Peristiwa Lambda Anda untuk membaca item. Untuk mempelajari cara mengonfigurasi streaming agar memungkinkan akses lintas akun, lihat Berbagi akses dengan AWS Lambda fungsi lintas akun di panduan Pengembang Amazon Kinesis Streams.

Setelah mengonfigurasi aliran Anda dengan kebijakan berbasis sumber daya yang memberikan izin yang diperlukan kepada Fungsi Peristiwa Lambda, buat pemetaan sumber peristiwa menggunakan salah satu metode yang dijelaskan di bagian sebelumnya.

Jika Anda memilih untuk membuat pemetaan sumber acara menggunakan konsol Lambda, tempelkan ARN aliran Anda langsung ke bidang input. Jika Anda ingin menentukan konsumen untuk streaming Anda, menempelkan ARN konsumen secara otomatis mengisi bidang aliran.

Mengonfigurasi respons batch sebagian dengan Kinesis Data Streams dan Lambda

Ketika menggunakan dan memproses data streaming dari sumber peristiwa, secara default Lambda memeriksa urutan nomor tertinggi batch hanya ketika batch berhasil diselesaikan. Lambda memperlakukan semua hasil lain sebagai sepenuhnya gagal dan mencoba lagi pemrosesan batch hingga batas coba lagi. Untuk memungkinkan keberhasilan parsial saat memproses batch dari aliran, aktifkan `ReportBatchItemFailures`. Mengizinkan keberhasilan sebagian dapat membantu mengurangi jumlah percobaan ulang pada catatan, meskipun itu tidak sepenuhnya mencegah kemungkinan percobaan ulang dalam catatan yang berhasil.

Untuk mengaktifkan `ReportBatchItemFailures`, masukkan nilai enum **`ReportBatchItemFailures`** dalam daftar FunctionResponseTypes. Daftar ini menunjukkan jenis respons mana yang diaktifkan untuk Fungsi Acara Anda. Anda dapat mengonfigurasi daftar ini saat membuat atau memperbarui pemetaan sumber peristiwa.

> **Note**
>
> Bahkan ketika kode fungsi Anda mengembalikan respons kegagalan sebagian batch, respons ini tidak diproses oleh Lambda kecuali `ReportBatchItemFailures` fitur tersebut diaktifkan secara eksplisit untuk pemetaan sumber acara Anda.

Sintaks laporan

Saat mengonfigurasi pelaporan pada kegagalan item batch, kelas `StreamsEventResponse` dikembalikan dengan daftar kegagalan item batch. Anda dapat menggunakan objek `StreamsEventResponse` untuk mengembalikan nomor urutan catatan gagal pertama dalam batch. Anda juga dapat membuat kelas kustom Anda sendiri menggunakan sintaks respons yang benar. Struktur JSON berikut menunjukkan sintaks respons yang diperlukan:

```
{
```

```
  "batchItemFailures": [
        {
            "itemIdentifier": "<SequenceNumber>"
        }
    ]
}
```

> **Note**
>
> Jika `batchItemFailures` array berisi beberapa item, Lambda menggunakan catatan dengan nomor urut terendah sebagai pos pemeriksaan. Lambda kemudian mencoba kembali semua catatan mulai dari pos pemeriksaan itu.

Status berhasil dan gagal

Lambda memperlakukan batch sebagai sepenuhnya berhasil jika Anda mengembalikan salah satu dari berikut:

- Daftar `batchItemFailure` kosong
- Daftar `batchItemFailure` nol
- `EventResponse` kosong
- `EventResponse` nol

Lambda memperlakukan batch sebagai sepenuhnya gagal jika Anda mengembalikan salah satu dari berikut:

- String `itemIdentifier` kosong
- `itemIdentifier` nol
- `itemIdentifier` dengan nama kunci yang buruk

Lambda mencoba kembali kegagalan berdasarkan strategi coba lagi Anda.

Membagi batch

Jika invokasi Anda gagal dan `BisectBatchOnFunctionError` diaktifkan, batch dibagi terlepas dari pengaturan `ReportBatchItemFailures` Anda.

Ketika respons berhasil batch parsial diterima dan kedua `BisectBatchOnFunctionError` dan `ReportBatchItemFailures` diaktifkan, batch dibagi dua di nomor urutan yang dikembalikan dan Lambda mencoba hanya catatan yang tersisa.

Untuk menyederhanakan implementasi logika respons batch sebagian, pertimbangkan untuk menggunakan utilitas Prosesor Batch dari Powertools untuk AWS Lambda, yang secara otomatis menangani kompleksitas ini untuk Anda.

Berikut adalah beberapa contoh kode fungsi yang mengembalikan daftar ID pesan yang gagal dalam batch:

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch Kinesis dengan Lambda menggunakan.NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text;
using System.Text.Json.Serialization;
using Amazon.Lambda.Core;
using Amazon.Lambda.KinesisEvents;
using AWS.Lambda.Powertools.Logging;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri

namespace KinesisIntegration;

public class Function
{
    // Powertools Logger requires an environment variables against your function
    // POWERTOOLS_SERVICE_NAME
```

```
   [Logging(LogEvent = true)]
   public async Task<StreamsEventResponse> FunctionHandler(KinesisEvent evnt,
ILambdaContext context)
   {
       if (evnt.Records.Count == 0)
       {
           Logger.LogInformation("Empty Kinesis Event received");
           return new StreamsEventResponse();
       }

       foreach (var record in evnt.Records)
       {
           try
           {
               Logger.LogInformation($"Processed Event with EventId:
{record.EventId}");
               string data = await GetRecordDataAsync(record.Kinesis, context);
               Logger.LogInformation($"Data: {data}");
               // TODO: Do interesting work based on the new data
           }
           catch (Exception ex)
           {
               Logger.LogError($"An error occurred {ex.Message}");
               /* Since we are working with streams, we can return the failed
item immediately.
                  Lambda will immediately begin to retry processing from this
failed item onwards. */
               return new StreamsEventResponse
               {
                   BatchItemFailures = new
List<StreamsEventResponse.BatchItemFailure>
                   {
                       new StreamsEventResponse.BatchItemFailure
{ ItemIdentifier = record.Kinesis.SequenceNumber }
                   }
               };
           }
       }
       Logger.LogInformation($"Successfully processed {evnt.Records.Count}
records.");
       return new StreamsEventResponse();
   }
```

```
    private async Task<string> GetRecordDataAsync(KinesisEvent.Record record,
 ILambdaContext context)
    {
        byte[] bytes = record.Data.ToArray();
        string data = Encoding.UTF8.GetString(bytes);
        await Task.CompletedTask; //Placeholder for actual async work
        return data;
    }
}

public class StreamsEventResponse
{
    [JsonPropertyName("batchItemFailures")]
    public IList<BatchItemFailure> BatchItemFailures { get; set; }
    public class BatchItemFailure
    {
        [JsonPropertyName("itemIdentifier")]
        public string ItemIdentifier { get; set; }
    }
}
```

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch Kinesis dengan Lambda menggunakan Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "fmt"
 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
```

```
)

func handler(ctx context.Context, kinesisEvent events.KinesisEvent)
 (map[string]interface{}, error) {
 batchItemFailures := []map[string]interface{}{}

 for _, record := range kinesisEvent.Records {
  curRecordSequenceNumber := ""

  // Process your record
  if /* Your record processing condition here */ {
   curRecordSequenceNumber = record.Kinesis.SequenceNumber
  }

  // Add a condition to check if the record processing failed
  if curRecordSequenceNumber != "" {
   batchItemFailures = append(batchItemFailures, map[string]interface{}
{"itemIdentifier": curRecordSequenceNumber})
  }
 }

 kinesisBatchResponse := map[string]interface{}{
  "batchItemFailures": batchItemFailures,
 }
 return kinesisBatchResponse, nil
}

func main() {
 lambda.Start(handler)
}
```

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch Kinesis dengan Lambda menggunakan Java.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.KinesisEvent;
import com.amazonaws.services.lambda.runtime.events.StreamsEventResponse;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

public class ProcessKinesisRecords implements RequestHandler<KinesisEvent,
 StreamsEventResponse> {

    @Override
    public StreamsEventResponse handleRequest(KinesisEvent input, Context
 context) {

        List<StreamsEventResponse.BatchItemFailure> batchItemFailures = new
 ArrayList<>();
        String curRecordSequenceNumber = "";

        for (KinesisEvent.KinesisEventRecord kinesisEventRecord :
 input.getRecords()) {
            try {
                //Process your record
                KinesisEvent.Record kinesisRecord =
 kinesisEventRecord.getKinesis();
                curRecordSequenceNumber = kinesisRecord.getSequenceNumber();

            } catch (Exception e) {
                /* Since we are working with streams, we can return the failed
 item immediately.
                   Lambda will immediately begin to retry processing from this
 failed item onwards. */
                batchItemFailures.add(new
 StreamsEventResponse.BatchItemFailure(curRecordSequenceNumber));
                return new StreamsEventResponse(batchItemFailures);
            }
        }

       return new StreamsEventResponse(batchItemFailures);
```

```
    }
}
```

JavaScript

SDK untuk JavaScript (v3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch Kinesis dengan Lambda menggunakan Javascript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const record of event.Records) {
    try {
      console.log(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      console.log(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      console.error(`An error occurred ${err}`);
      /* Since we are working with streams, we can return the failed item
 immediately.
           Lambda will immediately begin to retry processing from this failed
 item onwards. */
      return {
        batchItemFailures: [{ itemIdentifier: record.kinesis.sequenceNumber }],
      };
    }
  }
  console.log(`Successfully processed ${event.Records.length} records.`);
  return { batchItemFailures: [] };
};

async function getRecordDataAsync(payload) {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
```

```
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

Melaporkan kegagalan item batch Kinesis dengan penggunaan Lambda. TypeScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  KinesisStreamEvent,
  Context,
  KinesisStreamHandler,
  KinesisStreamRecordPayload,
  KinesisStreamBatchResponse,
} from "aws-lambda";
import { Buffer } from "buffer";
import { Logger } from "@aws-lambda-powertools/logger";

const logger = new Logger({
  logLevel: "INFO",
  serviceName: "kinesis-stream-handler-sample",
});

export const functionHandler: KinesisStreamHandler = async (
  event: KinesisStreamEvent,
  context: Context
): Promise<KinesisStreamBatchResponse> => {
  for (const record of event.Records) {
    try {
      logger.info(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      logger.info(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      logger.error(`An error occurred ${err}`);
      /* Since we are working with streams, we can return the failed item
 immediately.
           Lambda will immediately begin to retry processing from this failed
 item onwards. */
      return {
        batchItemFailures: [{ itemIdentifier: record.kinesis.sequenceNumber }],
      };
```

```
    }
  }
  logger.info(`Successfully processed ${event.Records.length} records.`);
  return { batchItemFailures: [] };
};

async function getRecordDataAsync(
  payload: KinesisStreamRecordPayload
): Promise<string> {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

PHP

SDK untuk PHP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch Kinesis dengan Lambda menggunakan PHP.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\Kinesis\KinesisEvent;
use Bref\Event\Handler as StdHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler implements StdHandler
{
```

```
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handle(mixed $event, Context $context): array
    {
        $kinesisEvent = new KinesisEvent($event);
        $this->logger->info("Processing records");
        $records = $kinesisEvent->getRecords();

        $failedRecords = [];
        foreach ($records as $record) {
            try {
                $data = $record->getData();
                $this->logger->info(json_encode($data));
                // TODO: Do interesting work based on the new data
            } catch (Exception $e) {
                $this->logger->error($e->getMessage());
                // failed processing the record
                $failedRecords[] = $record->getSequenceNumber();
            }
        }
        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords records");

        // change format for the response
        $failures = array_map(
            fn(string $sequenceNumber) => ['itemIdentifier' => $sequenceNumber],
            $failedRecords
        );

        return [
            'batchItemFailures' => $failures
        ];
    }
}

$logger = new StderrLogger();
```

```
return new Handler($logger);
```

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch Kinesis dengan Lambda menggunakan Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def handler(event, context):
    records = event.get("Records")
    curRecordSequenceNumber = ""

    for record in records:
        try:
            # Process your record
            curRecordSequenceNumber = record["kinesis"]["sequenceNumber"]
        except Exception as e:
            # Return failed record's sequence number
            return {"batchItemFailures":[{"itemIdentifier":
 curRecordSequenceNumber}]}

    return {"batchItemFailures":[]}
```

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch Kinesis dengan Lambda menggunakan Ruby.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
require 'aws-sdk'

def lambda_handler(event:, context:)
  batch_item_failures = []

  event['Records'].each do |record|
    begin
      puts "Processed Kinesis Event - EventID: #{record['eventID']}"
      record_data = get_record_data_async(record['kinesis'])
      puts "Record Data: #{record_data}"
      # TODO: Do interesting work based on the new data
    rescue StandardError => err
      puts "An error occurred #{err}"
      # Since we are working with streams, we can return the failed item
 immediately.
      # Lambda will immediately begin to retry processing from this failed item
 onwards.
      return { batchItemFailures: [{ itemIdentifier: record['kinesis']
['sequenceNumber'] }] }
    end
  end

  puts "Successfully processed #{event['Records'].length} records."
  { batchItemFailures: batch_item_failures }
end

def get_record_data_async(payload)
  data = Base64.decode64(payload['data']).force_encoding('utf-8')
  # Placeholder for actual async work
```

```
  sleep(1)
  data
end
```

Rust

SDK for Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch Kinesis dengan Lambda menggunakan Rust.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::{
    event::kinesis::KinesisEvent,
    kinesis::KinesisEventRecord,
    streams::{KinesisBatchItemFailure, KinesisEventResponse},
};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

async fn function_handler(event: LambdaEvent<KinesisEvent>) ->
 Result<KinesisEventResponse, Error> {
    let mut response = KinesisEventResponse {
        batch_item_failures: vec![],
    };

    if event.payload.records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(response);
    }

    for record in &event.payload.records {
        tracing::info!(
            "EventId: {}",
            record.event_id.as_deref().unwrap_or_default()
        );
```

```
        let record_processing_result = process_record(record);

        if record_processing_result.is_err() {
            response.batch_item_failures.push(KinesisBatchItemFailure {
                item_identifier: record.kinesis.sequence_number.clone(),
            });
            /* Since we are working with streams, we can return the failed item
 immediately.
            Lambda will immediately begin to retry processing from this failed
 item onwards. */
            return Ok(response);
        }
    }

    tracing::info!(
        "Successfully processed {} records",
        event.payload.records.len()
    );

    Ok(response)
}

fn process_record(record: &KinesisEventRecord) -> Result<(), Error> {
    let record_data = std::str::from_utf8(record.kinesis.data.as_slice());

    if let Some(err) = record_data.err() {
        tracing::error!("Error: {}", err);
        return Err(Error::from(err));
    }

    let record_data = record_data.unwrap_or_default();

    // do something interesting with the data
    tracing::info!("Data: {}", record_data);

    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        // disable printing the name of the module in every log line.
        .with_target(false)
```

```
        // disabling time is handy because CloudWatch will add the ingestion
 time.
        .without_time()
        .init();

    run(service_fn(function_handler)).await
}
```

Menggunakan Powertools untuk AWS Lambda prosesor batch

Utilitas prosesor batch dari Powertools untuk AWS Lambda secara otomatis menangani logika respons batch paral, mengurangi kompleksitas penerapan pelaporan kegagalan batch. Berikut adalah contoh menggunakan prosesor batch:

Python

> **Note**
>
> Untuk contoh lengkap dan petunjuk penyiapan, lihat dokumentasi prosesor batch.

Memproses Kinesis Data Streams stream AWS Lambda record dengan batch processor.

```
import json
from aws_lambda_powertools import Logger
from aws_lambda_powertools.utilities.batch import BatchProcessor, EventType,
 process_partial_response
from aws_lambda_powertools.utilities.data_classes import KinesisEvent
from aws_lambda_powertools.utilities.typing import LambdaContext

processor = BatchProcessor(event_type=EventType.KinesisDataStreams)
logger = Logger()

def record_handler(record):
    logger.info(record)
    # Your business logic here
    # Raise an exception to mark this record as failed

def lambda_handler(event, context: LambdaContext):
    return process_partial_response(
        event=event,
```

```
        record_handler=record_handler,
        processor=processor,
        context=context
    )
```

TypeScript

> **Note**
>
> Untuk contoh lengkap dan petunjuk penyiapan, lihat dokumentasi prosesor batch.

Memproses Kinesis Data Streams stream AWS Lambda record dengan batch processor.

```
import { BatchProcessor, EventType, processPartialResponse } from '@aws-lambda-
powertools/batch';
import { Logger } from '@aws-lambda-powertools/logger';
import type { KinesisEvent, Context } from 'aws-lambda';

const processor = new BatchProcessor(EventType.KinesisDataStreams);
const logger = new Logger();

const recordHandler = async (record: any): Promise<void> => {
    logger.info('Processing record', { record });
    // Your business logic here
    // Throw an error to mark this record as failed
};

export const handler = async (event: KinesisEvent, context: Context) => {
    return processPartialResponse(event, recordHandler, processor, {
        context,
    });
};
```

Java

> **Note**
>
> Untuk contoh lengkap dan petunjuk penyiapan, lihat dokumentasi prosesor batch.

Memproses Kinesis Data Streams stream AWS Lambda record dengan batch processor.

```
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.KinesisEvent;
import com.amazonaws.services.lambda.runtime.events.StreamsEventResponse;
import software.amazon.lambda.powertools.batch.BatchMessageHandlerBuilder;
import software.amazon.lambda.powertools.batch.handler.BatchMessageHandler;

public class KinesisStreamBatchHandler implements RequestHandler<KinesisEvent,
 StreamsEventResponse> {

    private final BatchMessageHandler<KinesisEvent, StreamsEventResponse> handler;

    public KinesisStreamBatchHandler() {
        handler = new BatchMessageHandlerBuilder()
                .withKinesisBatchHandler()
                .buildWithRawMessageHandler(this::processMessage);
    }

    @Override
    public StreamsEventResponse handleRequest(KinesisEvent kinesisEvent, Context
 context) {
        return handler.processBatch(kinesisEvent, context);
    }

    private void processMessage(KinesisEvent.KinesisEventRecord kinesisEventRecord,
 Context context) {
        // Process the stream record
    }
}
```

.NET

> **Note**
>
> Untuk contoh lengkap dan petunjuk penyiapan, lihat dokumentasi prosesor batch.

Memproses Kinesis Data Streams stream AWS Lambda record dengan batch processor.

```
using System;
using System.Threading;
using System.Threading.Tasks;
```

```
using Amazon.Lambda.Core;
using Amazon.Lambda.KinesisEvents;
using Amazon.Lambda.Serialization.SystemTextJson;
using AWS.Lambda.Powertools.BatchProcessing;

[assembly: LambdaSerializer(typeof(DefaultLambdaJsonSerializer))]

namespace HelloWorld;

public class OrderEvent
{
    public string? OrderId { get; set; }
    public string? CustomerId { get; set; }
    public decimal Amount { get; set; }
    public DateTime OrderDate { get; set; }
}

internal class TypedKinesisRecordHandler : ITypedRecordHandler<OrderEvent>
{
    public async Task<RecordHandlerResult> HandleAsync(OrderEvent orderEvent,
 CancellationToken cancellationToken)
    {
        if (string.IsNullOrEmpty(orderEvent.OrderId))
        {
            throw new ArgumentException("Order ID is required");
        }

        return await Task.FromResult(RecordHandlerResult.None);
    }
}

public class Function
{
    [BatchProcessor(TypedRecordHandler = typeof(TypedKinesisRecordHandler))]
    public BatchItemFailuresResponse HandlerUsingTypedAttribute(KinesisEvent _)
    {
        return TypedKinesisStreamBatchProcessor.Result.BatchItemFailuresResponse;
    }
}
```

Menyimpan catatan batch yang dibuang untuk sumber peristiwa Kinesis Data Streams di Lambda

Penanganan kesalahan untuk pemetaan sumber peristiwa Kinesis tergantung pada apakah kesalahan terjadi sebelum Fungsi Peristiwa dipanggil atau selama pemanggilan fungsi:

- Sebelum pemanggilan: Jika pemetaan sumber peristiwa Lambda tidak dapat menjalankan fungsi karena pelambatan atau masalah lain, ia akan mencoba lagi hingga catatan kedaluwarsa atau melebihi usia maksimum yang dikonfigurasi pada pemetaan sumber peristiwa (). MaximumRecordAgeInSeconds
- Selama pemanggilan: Jika fungsi dipanggil tetapi menampilkan kesalahan, Lambda mencoba ulang hingga catatan kedaluwarsa, melebihi usia maksimum (MaximumRecordAgeInSeconds), atau mencapai kuota coba ulang yang dikonfigurasi (). MaximumRetryAttempts Untuk kesalahan fungsi, Anda juga dapat mengonfigurasi BisectBatchOnFunctionError, yang membagi batch yang gagal menjadi dua batch yang lebih kecil, mengisolasi catatan buruk dan menghindari batas waktu. Memisahkan batch tidak menghabiskan kuota coba lagi.

Jika tindakan penanganan kesalahan gagal, Lambda membuang rekaman dan melanjutkan pemrosesan batch dari aliran. Dengan pengaturan default, ini berarti rekaman yang buruk dapat memblokir pemrosesan pada shard yang terpengaruh selama hingga satu minggu. Untuk menghindari hal ini, konfigurasikan pemetaan sumber kejadian fungsi Anda dengan jumlah percobaan ulang yang wajar dan usia maksimum rekaman yang sesuai dengan kasus penggunaan Anda.

Mengonfigurasi tujuan untuk pemanggilan yang gagal

Untuk menyimpan catatan pemanggilan pemetaan sumber peristiwa yang gagal, tambahkan tujuan ke pemetaan sumber peristiwa Fungsi Acara Anda. Setiap catatan yang dikirim ke tujuan adalah dokumen JSON yang berisi metadata tentang pemanggilan yang gagal. Untuk tujuan Amazon S3, Lambda juga mengirimkan seluruh catatan pemanggilan bersama dengan metadata. Anda dapat mengonfigurasi topik Amazon SNS, antrian Amazon SQS, bucket Amazon S3, atau Kafka sebagai tujuan.

Dengan tujuan Amazon S3, Anda dapat menggunakan fitur Pemberitahuan Acara Amazon S3 untuk menerima notifikasi saat objek diunggah ke bucket S3 tujuan Anda. Anda juga dapat mengonfigurasi Pemberitahuan Acara S3 untuk memanggil Fungsi Peristiwa Lambda lain untuk melakukan pemrosesan otomatis pada batch yang gagal.

Peran eksekusi Anda harus memiliki izin untuk tujuan:

- Untuk tujuan SQS: sqs: SendMessage
- Untuk tujuan SNS: SNS: Publish
- Untuk tujuan S3: s3: PutObject dan s3: ListBucket
- Untuk tujuan Kafka: kafka-cluster: WriteData

Anda dapat mengonfigurasi topik Kafka sebagai tujuan gagal untuk pemetaan sumber acara Kafka Anda. Ketika Lambda tidak dapat memproses catatan setelah percobaan ulang yang melelahkan atau ketika catatan melebihi usia maksimum, Lambda mengirimkan catatan yang gagal ke topik Kafka yang ditentukan untuk diproses nanti. Lihat the section called “Tujuan kegagalan Kafka”.

Jika Anda telah mengaktifkan enkripsi dengan kunci KMS Anda sendiri untuk tujuan S3, peran eksekusi fungsi Anda juga harus memiliki izin untuk memanggil kms:. GenerateDataKey Jika kunci KMS dan tujuan bucket S3 berada di akun yang berbeda dari Fungsi Peristiwa Lambda dan peran eksekusi, konfigurasikan kunci KMS untuk mempercayai peran eksekusi agar memungkinkan kms:. GenerateDataKey

Untuk mengonfigurasi tujuan saat gagal menggunakan konsol, ikuti langkah-langkah berikut:

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih fungsi.
3. Di bagian Gambaran umum fungsi, pilih Tambahkan tujuan.
4. Untuk Sumber, pilih Pemanggilan pemetaan sumber acara.
5. Untuk pemetaan sumber peristiwa, pilih sumber peristiwa yang dikonfigurasi untuk fungsi ini.
6. Untuk Kondisi, pilih On failure. Untuk pemanggilan pemetaan sumber peristiwa, ini adalah satu-satunya kondisi yang diterima.
7. Untuk tipe Tujuan, pilih tipe tujuan yang Lambda kirimkan catatan pemanggilan.
8. Untuk Tujuan, pilih sumber daya.
9. Pilih Simpan.

Anda juga dapat mengonfigurasi tujuan pada kegagalan menggunakan AWS Command Line Interface (AWS CLI). Misalnya, perintah create-event-source-mapping berikut menambahkan pemetaan sumber peristiwa dengan tujuan kegagalan SQS ke: `MyFunction`

```
aws lambda create-event-source-mapping \
--function-name "MyFunction" \
--event-source-arn arn:aws:kinesis:us-east-2:123456789012:stream/lambda-stream \
--destination-config '{"OnFailure": {"Destination": "arn:aws:sqs:us-
east-1:123456789012:dest-queue"}}'
```

Perintah update-event-source-mapping berikut memperbarui pemetaan sumber peristiwa untuk mengirim catatan pemanggilan yang gagal ke tujuan SNS setelah dua percobaan ulang, atau jika catatan berusia lebih dari satu jam.

```
aws lambda update-event-source-mapping \
--uuid f89f8514-cdd9-4602-9e1f-01a5b77d449b \
--maximum-retry-attempts 2 \
--maximum-record-age-in-seconds 3600 \
--destination-config '{"OnFailure": {"Destination": "arn:aws:sns:us-
east-1:123456789012:dest-topic"}}'
```

Pengaturan yang diperbarui diterapkan secara asinkron dan tidak muncul dalam output hingga proses selesai. Gunakan perintah get-event-source-mapping untuk melihat status saat ini.

Untuk menghapus tujuan, berikan string kosong sebagai argumen ke `destination-config` parameter:

```
aws lambda update-event-source-mapping \
--uuid f89f8514-cdd9-4602-9e1f-01a5b77d449b \
--destination-config '{"OnFailure": {"Destination": ""}}'
```

Praktik terbaik keamanan untuk tujuan Amazon S3

Menghapus bucket S3 yang dikonfigurasi sebagai tujuan tanpa menghapus tujuan dari konfigurasi fungsi Anda dapat menimbulkan risiko keamanan. Jika pengguna lain mengetahui nama bucket tujuan Anda, mereka dapat membuat ulang bucket di bucket tersebut. Akun AWS Catatan pemanggilan yang gagal akan dikirim ke bucket mereka, yang berpotensi mengekspos data dari fungsi Anda.

> **Warning**
>
> Untuk memastikan bahwa catatan pemanggilan dari Fungsi Peristiwa Anda tidak dapat dikirim ke bucket S3 di bucket lain Akun AWS, tambahkan kondisi ke peran eksekusi fungsi Anda yang membatasi `s3:PutObject` izin ke bucket di akun Anda.

Contoh berikut menunjukkan kebijakan IAM yang membatasi `s3:PutObject` izin fungsi Anda ke bucket di akun Anda. Kebijakan ini juga memberi Lambda `s3:ListBucket` izin yang dibutuhkan untuk menggunakan bucket S3 sebagai tujuan.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "S3BucketResourceAccountWrite",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::*/*",
                "arn:aws:s3:::*"
            ],
            "Condition": {
                "StringEquals": {
                    "s3:ResourceAccount": "111122223333"
                }
            }
        }
    ]
}
```

Untuk menambahkan kebijakan izin ke peran eksekusi fungsi Anda menggunakan Konsol Manajemen AWS atau AWS CLI, lihat instruksi dalam prosedur berikut:

Console

Untuk menambahkan kebijakan izin ke peran eksekusi fungsi (konsol)

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih Lambda Event Function yang peran eksekusinya ingin Anda ubah.
3. Di tab Konfigurasi, pilih Izin.
4. Di tab Peran eksekusi, pilih nama Peran fungsi Anda untuk membuka halaman konsol IAM peran.
5. Tambahkan kebijakan izin ke peran dengan melakukan hal berikut:
   a. Di panel Kebijakan izin, pilih Tambahkan izin, lalu pilih Buat kebijakan sebaris.
   b. Di Editor kebijakan, pilih JSON.
   c. Rekatkan kebijakan yang ingin Anda tambahkan ke editor (ganti JSON yang ada), lalu pilih Berikutnya.
   d. Di bawah Detail kebijakan, masukkan nama Kebijakan.
   e. Pilih Buat kebijakan.

AWS CLI

Untuk menambahkan kebijakan izin ke peran eksekusi Fungsi Peristiwa (CLI)

1. Buat dokumen kebijakan JSON dengan izin yang diperlukan dan simpan di direktori lokal.
2. Gunakan perintah IAM `put-role-policy` CLI untuk menambahkan izin ke peran eksekusi fungsi Anda. Jalankan perintah berikut dari direktori tempat Anda menyimpan dokumen kebijakan JSON dan ganti nama peran, nama kebijakan, dan dokumen kebijakan dengan nilai Anda sendiri.

```
aws iam put-role-policy \
--role-name my_lambda_role \
--policy-name LambdaS3DestinationPolicy \
--policy-document file://my_policy.json
```

Contoh catatan pemanggilan Amazon SNS dan Amazon SQS

Contoh berikut menunjukkan apa yang Lambda kirim ke antrian SQS atau topik SNS untuk pemanggilan sumber peristiwa Kinesis yang gagal. Karena Lambda hanya mengirimkan metadata untuk jenis tujuan ini, gunakan,, `streamArn` `shardIdstartSequenceNumber`, dan `endSequenceNumber` bidang untuk mendapatkan catatan asli lengkap. Semua bidang yang ditampilkan di `KinesisBatchInfo` properti akan selalu ada.

```
{
    "requestContext": {
        "requestId": "c9b8fa9f-5a7f-xmpl-af9c-0c604cde93a5",
        "functionArn": "arn:aws:lambda:us-east-2:123456789012:function:myfunction",
        "condition": "RetryAttemptsExhausted",
        "approximateInvokeCount": 1
    },
    "responseContext": {
        "statusCode": 200,
        "executedVersion": "$LATEST",
        "functionError": "Unhandled"
    },
    "version": "1.0",
    "timestamp": "2019-11-14T00:38:06.021Z",
    "KinesisBatchInfo": {
        "shardId": "shardId-000000000001",
        "startSequenceNumber":
 "49601189658422359378836298521827638475320189012309704722",
        "endSequenceNumber":
 "49601189658422359378836298522902373528957594348623495186",
        "approximateArrivalOfFirstRecord": "2019-11-14T00:38:04.835Z",
        "approximateArrivalOfLastRecord": "2019-11-14T00:38:05.580Z",
        "batchSize": 500,
        "streamArn": "arn:aws:kinesis:us-east-2:123456789012:stream/mystream"
    }
}
```

Anda dapat menggunakan informasi ini guna mengambil rekaman yang terpengaruh dari aliran untuk pemecahan masalah. Rekaman aktual tidak disertakan, jadi Anda harus memproses rekaman ini dan mengambilnya dari aliran sebelum kedaluwarsa dan hilang.

Contoh catatan pemanggilan Amazon S3

Contoh berikut menunjukkan apa yang dikirim Lambda ke bucket Amazon S3 untuk pemanggilan sumber peristiwa Kinesis yang gagal. Selain semua bidang dari contoh sebelumnya untuk tujuan SQS dan SNS, `payload` bidang berisi catatan pemanggilan asli sebagai string JSON yang lolos.

```
{
    "requestContext": {
        "requestId": "c9b8fa9f-5a7f-xmpl-af9c-0c604cde93a5",
        "functionArn": "arn:aws:lambda:us-east-2:123456789012:function:myfunction",
        "condition": "RetryAttemptsExhausted",
        "approximateInvokeCount": 1
    },
    "responseContext": {
        "statusCode": 200,
        "executedVersion": "$LATEST",
        "functionError": "Unhandled"
    },
    "version": "1.0",
    "timestamp": "2019-11-14T00:38:06.021Z",
    "KinesisBatchInfo": {
        "shardId": "shardId-000000000001",
        "startSequenceNumber":
 "49601189658422359378836298521827638475320189012309704722",
        "endSequenceNumber":
 "49601189658422359378836298522902373528957594348623495186",
        "approximateArrivalOfFirstRecord": "2019-11-14T00:38:04.835Z",
        "approximateArrivalOfLastRecord": "2019-11-14T00:38:05.580Z",
        "batchSize": 500,
        "streamArn": "arn:aws:kinesis:us-east-2:123456789012:stream/mystream"
    },
    "payload": "<Whole Event>" // Only available in S3
}
```

Objek S3 yang berisi catatan pemanggilan menggunakan konvensi penamaan berikut:

```
aws/lambda/<ESM-UUID>/<shardID>/YYYY/MM/DD/YYYY-MM-DDTHH.MM.SS-<Random UUID>
```

Menerapkan pemrosesan Kinesis Data Streams stateful di Lambda

Lambda Event Functions dapat menjalankan aplikasi pemrosesan aliran berkelanjutan. Aliran merupakan data tidak terbatas yang mengalir terus-menerus melalui aplikasi Anda. Untuk

menganalisis informasi dari input yang terus diperbarui ini, Anda dapat mengikat catatan yang disertakan menggunakan jendela yang didefinisikan dalam hal waktu.

Jatuh jendela adalah jendela waktu yang berbeda yang membuka dan menutup secara berkala. Secara default, pemanggilan Lambda tidak memiliki status — Anda tidak dapat menggunakannya untuk memproses data di beberapa pemanggilan berkelanjutan tanpa database eksternal. Namun, dengan jendela yang jatuh, Anda dapat mempertahankan status Anda di seluruh pemanggilan.

Status ini berisi hasil agregat pesan yang sebelumnya diproses untuk jendela saat ini. Status Anda bisa maksimal 1 MB untuk setiap pecahan. Jika melebihi ukuran tersebut, Lambda mengakhiri jendela lebih awal.

Setiap rekaman dalam aliran milik jendela tertentu. Lambda memproses setiap rekaman setidaknya sekali, tetapi tidak menjamin bahwa setiap catatan diproses hanya sekali. Dalam kasus yang jarang terjadi, seperti penanganan kesalahan, beberapa catatan mungkin diproses lebih dari sekali. Catatan selalu diproses secara berurutan pertama kali. Jika catatan diproses lebih dari satu kali, mereka mungkin diproses rusak.

Agregasi dan pemrosesan

Fungsi yang dikelola pengguna Anda dipanggil baik untuk agregasi maupun untuk memproses hasil akhir agregasi tersebut. Lambda mengumpulkan semua catatan yang diterima di jendela. Anda dapat menerima catatan ini dalam beberapa batch, masing-masing sebagai invokasi terpisah. Setiap invokasi menerima status.

Jadi, saat menggunakan jendela tumbling, respons Fungsi Peristiwa Lambda Anda harus berisi `state` properti. Jika respons tidak berisi `state` properti, Lambda menganggap ini sebagai pemanggilan yang gagal. Untuk memenuhi kondisi ini, fungsi Anda dapat mengembalikan `TimeWindowEventResponse` objek, yang memiliki bentuk JSON berikut:

Example Nilai **`TimeWindowEventResponse`**

```
{
    "state": {
        "1": 282,
        "2": 715
    },
    "batchItemFailures": []
}
```

> **Note**
>
> Untuk fungsi Java, sebaiknya gunakan a `Map<String, String>` untuk mewakili status.

Di akhir jendela, bendera `isFinalInvokeForWindow` diatur `true` untuk menunjukkan bahwa ini adalah status akhir dan siap untuk diproses. Setelah pemrosesan, jendela selesai, dan invokasi akhir Anda selesai, lalu status dihapus.

Di akhir jendela Anda, Lambda menggunakan pemrosesan akhir untuk tindakan pada hasil agregasi. Pemrosesan akhir Anda dipanggil secara sinkron. Setelah pemanggilan berhasil, fungsi Anda memeriksa nomor urut dan pemrosesan aliran berlanjut. Jika pemanggilan tidak berhasil, Fungsi Acara Lambda Anda menangguhkan pemrosesan lebih lanjut hingga pemanggilan berhasil.

Example KinesisTimeWindowEvent

```
{
    "Records": [
        {
            "kinesis": {
                "kinesisSchemaVersion": "1.0",
                "partitionKey": "1",
                "sequenceNumber":
 "49590338271490256608559692538361571095921575989136588898",
                "data": "SGVsbG8sIHRoaXMgaXMgYSB0ZXN0Lg==",
                "approximateArrivalTimestamp": 1607497475.000
            },
            "eventSource": "aws:kinesis",
            "eventVersion": "1.0",
            "eventID":
 "shardId-000000000006:49590338271490256608559692538361571095921575989136588898",
            "eventName": "aws:kinesis:record",
            "invokeIdentityArn": "arn:aws:iam::123456789012:role/lambda-kinesis-role",
            "awsRegion": "us-east-1",
            "eventSourceARN": "arn:aws:kinesis:us-east-1:123456789012:stream/lambda-
stream"
        }
    ],
    "window": {
        "start": "2020-12-09T07:04:00Z",
        "end": "2020-12-09T07:06:00Z"
```

```
    },
    "state": {
        "1": 282,
        "2": 715
    },
    "shardId": "shardId-000000000006",
    "eventSourceARN": "arn:aws:kinesis:us-east-1:123456789012:stream/lambda-stream",
    "isFinalInvokeForWindow": false,
    "isWindowTerminatedEarly": false
}
```

Konfigurasi

Anda dapat mengonfigurasi jendela berguling saat membuat atau memperbarui pemetaan sumber peristiwa. Untuk mengkonfigurasi jendela tumbling, tentukan jendela dalam hitungan detik (TumblingWindowInSeconds). Contoh berikut AWS Command Line Interface (AWS CLI) perintah membuat pemetaan sumber acara streaming yang memiliki jendela jatuh 120 detik. Fungsi Peristiwa Lambda yang didefinisikan untuk agregasi dan pemrosesan diberi nama. `tumbling-window-example-function`

```
aws lambda create-event-source-mapping \
--event-source-arn arn:aws:kinesis:us-east-1:123456789012:stream/lambda-stream \
--function-name tumbling-window-example-function \
--starting-position TRIM_HORIZON \
--tumbling-window-in-seconds 120
```

Lambda menentukan jatuh batas jendela berguling berdasarkan waktu ketika catatan dimasukkan ke dalam aliran. Semua catatan memiliki stempel waktu perkiraan yang tersedia yang digunakan Lambda dalam penentuan batas.

Agregasi jendela berguling tidak mendukung shard ulang. Ketika pecahan berakhir, Lambda menganggap jendela saat ini ditutup, dan setiap pecahan anak memulai jendela mereka sendiri dalam keadaan baru. Ketika tidak ada catatan baru yang ditambahkan ke jendela saat ini, Lambda menunggu hingga 2 menit sebelum mengasumsikan bahwa jendela sudah selesai. Ini membantu memastikan bahwa fungsi membaca semua catatan di jendela saat ini, bahkan jika catatan ditambahkan sebentar-sebentar.

Jendela berguling sepenuhnya mendukung kebijakan coba lagi yang ada `maxRetryAttempts` dan `maxRecordAge`.

Example Handler.py — Agregasi dan pemrosesan

Fungsi Python berikut menunjukkan cara untuk menggabungkan, lalu memproses status akhir Anda:

```
def lambda_handler(event, context):
    print('Incoming event: ', event)
    print('Incoming state: ', event['state'])

#Check if this is the end of the window to either aggregate or process.
    if event['isFinalInvokeForWindow']:
        # logic to handle final state of the window
        print('Destination invoke')
    else:
        print('Aggregate invoke')

#Check for early terminations
    if event['isWindowTerminatedEarly']:
        print('Window terminated early')

    #Aggregation logic
    state = event['state']
    for record in event['Records']:
        state[record['kinesis']['partitionKey']] = state.get(record['kinesis']
['partitionKey'], 0) + 1

    print('Returning state: ', state)
    return {'state': state}
```

Parameter Lambda untuk pemetaan sumber peristiwa Amazon Kinesis Data Streams

Semua pemetaan sumber peristiwa Lambda berbagi operasi yang sama CreateEventSourceMappingdan API. UpdateEventSourceMapping Namun, hanya beberapa parameter yang berlaku untuk Kinesis.

Parameter	Diperlukan	Default	Catatan
BatchSize	T	100	Maksimum: 10.000.
BisectBatchOnFunctionError	T	false	none

Parameter	Diperlukan	Default	Catatan
DestinationConfig	T	N/A	Antrian Amazon SQS atau tujuan topik Amazon SNS untuk catatan yang dibuang. Untuk informasi selengkapnya, lihat Mengonfigurasi tujuan untuk pemanggilan yang gagal.
Diaktifkan	T	true	none
EventSourceArn	Y	N/A	ARN dari aliran data atau konsumen aliran
FunctionName	Y	N/A	none
FunctionResponseTypes	T	N/A	Untuk membiarkan Fungsi Acara melaporkan kegagalan tertentu dalam batch, sertakan nilainya `ReportBatchItemFailures FunctionResponseTypes` . Untuk informasi selengkapnya, lihat Mengonfigurasi respons batch sebagian dengan Kinesis Data Streams dan Lambda.
MaximumBatchingWindowInSeconds	T	0	none

Parameter	Diperlukan	Default	Catatan
MaximumRecordAgeInSeconds	T	-1	-1 berarti tak terbatas: Lambda tidak membuang catatan (pengaturan retensi data Kinesis Data Streams masih berlaku) Minimal: -1 Maksimal: 604.800
MaximumRetryAttempts	T	-1	-1 berarti tak terbatas: catatan gagal dicoba ulang sampai catatan kedaluwarsa Minimal: -1 Maksimum: 10.000.
ParallelizationFactor	T	1	Maksimal: 10
StartingPosition	Y	N/A	AT_TIMESTAMP, TRIM_HORIZON, atau TERBARU
StartingPositionTimestamp	T	N/A	Hanya valid jika StartingPosition disetel ke AT_TIMESTAMP. Waktu untuk mulai membaca, dalam detik waktu Unix

Parameter	Diperlukan	Default	Catatan
TumblingWindowInSeconds	T	N/A	Minimal: 0 Maksimal: 900

Menggunakan pemfilteran acara dengan sumber acara Kinesis

Anda dapat menggunakan pemfilteran peristiwa untuk mengontrol rekaman mana dari aliran atau antrian yang dikirim Lambda ke Fungsi Acara Anda. Untuk informasi umum tentang cara kerja penyaringan acara, lihatthe section called “Pemfilteran peristiwa”.

Bagian ini berfokus pada penyaringan acara untuk sumber acara Kinesis.

> **Note**
>
> Pemetaan sumber peristiwa Kinesis hanya mendukung pemfilteran pada kunci. `data`

Topik

- Dasar-dasar penyaringan acara Kinesis
- Menyaring catatan agregat Kinesis

Dasar-dasar penyaringan acara Kinesis

Misalkan produser memasukkan data yang diformat JSON ke dalam aliran data Kinesis Anda. Contoh catatan akan terlihat seperti berikut, dengan data JSON dikonversi ke string yang dikodekan Base64 di lapangan. `data`

```
{
    "kinesis": {
        "kinesisSchemaVersion": "1.0",
        "partitionKey": "1",
        "sequenceNumber": "49590338271490256608559692538361571095921575989136588898",
        "data":
 "eyJSZWNvcmROdW1iZXIiOiAiMDAwMSIsICJUaW1lU3RhbXAiOiAieXl5eS1tbS1kZFRoaDptbTpzcyIsICJSZXF1ZXN0Q
        "approximateArrivalTimestamp": 1545084650.987
        },
```

```
    "eventSource": "aws:kinesis",
    "eventVersion": "1.0",
    "eventID":
 "shardId-000000000006:49590338271490256608559692538361571095921575989136588898",
    "eventName": "aws:kinesis:record",
    "invokeIdentityArn": "arn:aws:iam::123456789012:role/lambda-role",
    "awsRegion": "us-east-2",
    "eventSourceARN": "arn:aws:kinesis:us-east-2:123456789012:stream/lambda-stream"
}
```

Selama data yang dimasukkan produsen ke dalam aliran adalah JSON yang valid, Anda dapat menggunakan pemfilteran peristiwa untuk memfilter catatan menggunakan kunci. `data` Misalkan produser memasukkan catatan ke dalam aliran Kinesis Anda dalam format JSON berikut.

```
{
    "record": 12345,
    "order": {
        "type": "buy",
        "stock": "ANYCO",
        "quantity": 1000
        }
}
```

Untuk memfilter hanya catatan di mana jenis pesanan adalah “beli,” `FilterCriteria` objeknya adalah sebagai berikut.

```
{
    "Filters": [
        {
            "Pattern": "{ \"data\" : { \"order\" : { \"type\" : [ \"buy\" ] } } }"
        }
    ]
}
```

Untuk kejelasan tambahan, berikut adalah nilai filter yang `Pattern` diperluas di JSON biasa.

```
{
    "data": {
        "order": {
            "type": [ "buy" ]
            }
```

```
      }
}
```

Anda dapat menambahkan filter menggunakan konsol, AWS CLI atau AWS SAM templat.

Console

Untuk menambahkan filter ini menggunakan konsol, ikuti instruksi Melampirkan kriteria filter ke pemetaan sumber peristiwa (konsol) dan masukkan string berikut untuk kriteria Filter.

```
{ "data" : { "order" : { "type" : [ "buy" ] } } }
```

AWS CLI

Untuk membuat pemetaan sumber peristiwa baru dengan kriteria filter ini menggunakan AWS Command Line Interface (AWS CLI), jalankan perintah berikut.

```
aws lambda create-event-source-mapping \
    --function-name my-function \
    --event-source-arn arn:aws:kinesis:us-east-2:123456789012:stream/my-stream \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"data\" : { \"order\" : { \"type
\" : [ \"buy\" ] } } }"}]}'
```

Untuk menambahkan kriteria filter ini ke pemetaan sumber peristiwa yang ada, jalankan perintah berikut.

```
aws lambda update-event-source-mapping \
    --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"data\" : { \"order\" : { \"type
\" : [ \"buy\" ] } } }"}]}'
```

AWS SAM

Untuk menambahkan filter ini menggunakan AWS SAM, tambahkan cuplikan berikut ke template YAMAL untuk sumber acara Anda.

```
FilterCriteria:
  Filters:
    - Pattern: '{ "data" : { "order" : { "type" : [ "buy" ] } } }'
```

Untuk memfilter peristiwa dengan benar dari sumber Kinesis, bidang data dan kriteria filter Anda untuk bidang data harus dalam format JSON yang valid. Jika salah satu bidang tidak dalam format JSON yang valid, Lambda akan menghapus pesan atau melempar pengecualian. Tabel berikut merangkum perilaku spesifik:

Format data masuk	Format pola filter untuk properti data	Tindakan yang dihasilkan
JSON yang valid	JSON yang valid	Filter Lambda berdasarkan kriteria filter Anda.
JSON yang valid	Tidak ada pola filter untuk properti data	Filter Lambda (hanya pada properti metadata lainnya) berdasarkan kriteria filter Anda.
JSON yang valid	Non-JSON	Lambda melempar pengecualian pada saat pembuatan atau pembaruan pemetaan sumber acara. Pola filter untuk properti data harus dalam format JSON yang valid.
Non-JSON	JSON yang valid	Lambda menjatuhkan rekor.
Non-JSON	Tidak ada pola filter untuk properti data	Filter Lambda (hanya pada properti metadata lainnya) berdasarkan kriteria filter Anda.
Non-JSON	Non-JSON	Lambda melempar pengecualian pada saat pembuatan atau pembaruan pemetaan sumber acara. Pola filter untuk properti data harus dalam format JSON yang valid.

Menyaring catatan agregat Kinesis

Dengan Kinesis, Anda dapat menggabungkan beberapa catatan ke dalam satu catatan Kinesis Data Streams untuk meningkatkan throughput data Anda. Lambda hanya dapat menerapkan kriteria filter ke rekaman agregat saat Anda menggunakan Kinesis yang ditingkatkan fan-out. Memfilter rekaman agregat dengan Kinesis standar tidak didukung. Saat menggunakan fan-out yang disempurnakan, Anda mengonfigurasi konsumen throughput khusus Kinesis untuk bertindak sebagai pemicu Fungsi Peristiwa Lambda Anda. Lambda kemudian memfilter catatan agregat dan hanya meneruskan catatan yang memenuhi kriteria filter Anda.

Untuk mempelajari lebih lanjut tentang agregasi catatan Kinesis, lihat bagian Agregasi pada halaman Konsep Kunci Perpustakaan Produsen Kinesis (KPL). Untuk mempelajari lebih lanjut tentang menggunakan Lambda dengan Kinesis yang ditingkatkan fan-out, lihat Meningkatkan performa pemrosesan streaming real-time dengan Amazon Kinesis Data Streams yang disempurnakan oleh fan-out dan Lambda di blog komputasi. AWS AWS

Tutorial: Menggunakan Lambda dengan Kinesis Data Streams

Dalam tutorial ini, Anda membuat Fungsi Peristiwa Lambda untuk mengkonsumsi peristiwa dari aliran data Amazon Kinesis.

1. Aplikasi kustom menulis rekaman ke aliran.
2. AWS Lambda polling aliran dan, ketika mendeteksi catatan baru dalam aliran, memanggil Fungsi Peristiwa Lambda Anda.
3. AWS Lambda menjalankan Fungsi Peristiwa Lambda dengan mengasumsikan peran eksekusi yang Anda tentukan pada saat Anda membuat Fungsi Peristiwa Lambda.

Prasyarat

Instal AWS Command Line Interface

Jika Anda belum menginstal AWS Command Line Interface, ikuti langkah-langkah di Menginstal atau memperbarui versi terbaru AWS CLI untuk menginstalnya.

Tutorial ini membutuhkan terminal baris perintah atau shell untuk menjalankan perintah. Di Linux dan macOS, gunakan shell dan manajer paket pilihan Anda.

> **Note**
>
> Di Windows, beberapa perintah Bash CLI yang biasa Anda gunakan dengan Lambda (`zip`seperti) tidak didukung oleh terminal bawaan sistem operasi. Untuk mendapatkan Windows-integrated versi Ubuntu dan Bash, instal Subsistem Windows untuk Linux.

Buat peran eksekusi

Buat peran eksekusi yang memberikan izin fungsi Anda untuk mengakses AWS sumber daya.

Untuk membuat peran eksekusi

1. Buka halaman peran di konsol IAM.
2. Pilih Buat peran.
3. Buat peran dengan properti berikut.
   - Entitas tepercaya – AWS Lambda.
   - Izin – AWSLambdaKinesisExecutionRole.
   - Nama peran – **`lambda-kinesis-role`**.

AWSLambdaKinesisExecutionRoleKebijakan memiliki izin yang diperlukan fungsi untuk membaca item dari Kinesis dan menulis log ke Log CloudWatch .

Buat Fungsi Acara

Buat Fungsi Peristiwa Lambda yang memproses pesan Kinesis Anda. Kode fungsi mencatat ID peristiwa dan data peristiwa dari catatan Kinesis ke CloudWatch Log.

Tutorial ini menggunakan runtime Node.js 24, tetapi kami juga menyediakan kode contoh dalam bahasa runtime lainnya. Anda dapat memilih tab di kotak berikut untuk melihat kode runtime yang Anda minati. JavaScript Kode yang akan Anda gunakan dalam langkah ini adalah pada contoh pertama yang ditunjukkan di JavaScripttab.

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara Kinesis dengan Lambda menggunakan.NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.KinesisEvents;
using AWS.Lambda.Powertools.Logging;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri

namespace KinesisIntegrationSampleCode;

public class Function
{
    // Powertools Logger requires an environment variables against your function
    // POWERTOOLS_SERVICE_NAME
    [Logging(LogEvent = true)]
    public async Task FunctionHandler(KinesisEvent evnt, ILambdaContext context)
    {
        if (evnt.Records.Count == 0)
        {
            Logger.LogInformation("Empty Kinesis Event received");
            return;
        }

        foreach (var record in evnt.Records)
        {
            try
            {
```

```
                Logger.LogInformation($"Processed Event with EventId:
 {record.EventId}");
                string data = await GetRecordDataAsync(record.Kinesis, context);
                Logger.LogInformation($"Data: {data}");
                // TODO: Do interesting work based on the new data
            }
            catch (Exception ex)
            {
                Logger.LogError($"An error occurred {ex.Message}");
                throw;
            }
        }
        Logger.LogInformation($"Successfully processed {evnt.Records.Count}
 records.");
    }

    private async Task<string> GetRecordDataAsync(KinesisEvent.Record record,
 ILambdaContext context)
    {
        byte[] bytes = record.Data.ToArray();
        string data = Encoding.UTF8.GetString(bytes);
        await Task.CompletedTask; //Placeholder for actual async work
        return data;
    }
}
```

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara Kinesis dengan Lambda menggunakan Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main
```

```
import (
 "context"
 "log"

 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

func handler(ctx context.Context, kinesisEvent events.KinesisEvent) error {
 if len(kinesisEvent.Records) == 0 {
  log.Printf("empty Kinesis event received")
  return nil
 }

 for _, record := range kinesisEvent.Records {
  log.Printf("processed Kinesis event with EventId: %v", record.EventID)
  recordDataBytes := record.Kinesis.Data
  recordDataText := string(recordDataBytes)
  log.Printf("record data: %v", recordDataText)
  // TODO: Do interesting work based on the new data
 }
 log.Printf("successfully processed %v records", len(kinesisEvent.Records))
 return nil
}

func main() {
 lambda.Start(handler)
}
```

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara Kinesis dengan Lambda menggunakan Java.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.LambdaLogger;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.KinesisEvent;

public class Handler implements RequestHandler<KinesisEvent, Void> {
    @Override
    public Void handleRequest(final KinesisEvent event, final Context context) {
        LambdaLogger logger = context.getLogger();
        if (event.getRecords().isEmpty()) {
            logger.log("Empty Kinesis Event received");
            return null;
        }
        for (KinesisEvent.KinesisEventRecord record : event.getRecords()) {
            try {
                logger.log("Processed Event with EventId: "+record.getEventID());
                String data = new String(record.getKinesis().getData().array());
                logger.log("Data:"+ data);
                // TODO: Do interesting work based on the new data
            }
            catch (Exception ex) {
                logger.log("An error occurred:"+ex.getMessage());
                throw ex;
            }
        }
        logger.log("Successfully processed:"+event.getRecords().size()+"
 records");
        return null;
    }

}
```

JavaScript

SDK untuk JavaScript (v3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara Kinesis dengan menggunakan Lambda. JavaScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const record of event.Records) {
    try {
      console.log(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      console.log(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      console.error(`An error occurred ${err}`);
      throw err;
    }
  }
  console.log(`Successfully processed ${event.Records.length} records.`);
};

async function getRecordDataAsync(payload) {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

Mengkonsumsi acara Kinesis dengan menggunakan Lambda. TypeScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  KinesisStreamEvent,
```

```
  Context,
  KinesisStreamHandler,
  KinesisStreamRecordPayload,
} from "aws-lambda";
import { Buffer } from "buffer";
import { Logger } from "@aws-lambda-powertools/logger";

const logger = new Logger({
  logLevel: "INFO",
  serviceName: "kinesis-stream-handler-sample",
});

export const functionHandler: KinesisStreamHandler = async (
  event: KinesisStreamEvent,
  context: Context
): Promise<void> => {
  for (const record of event.Records) {
    try {
      logger.info(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      logger.info(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      logger.error(`An error occurred ${err}`);
      throw err;
    }
    logger.info(`Successfully processed ${event.Records.length} records.`);
  }
};

async function getRecordDataAsync(
  payload: KinesisStreamRecordPayload
): Promise<string> {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

PHP

SDK untuk PHP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara Kinesis dengan Lambda menggunakan PHP.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\Kinesis\KinesisEvent;
use Bref\Event\Kinesis\KinesisHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler extends KinesisHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handleKinesis(KinesisEvent $event, Context $context): void
    {
        $this->logger->info("Processing records");
        $records = $event->getRecords();
        foreach ($records as $record) {
            $data = $record->getData();
```

```
            $this->logger->info(json_encode($data));
            // TODO: Do interesting work based on the new data

            // Any exception thrown will be logged and the invocation will be
 marked as failed
        }
        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords records");
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara Kinesis dengan Lambda menggunakan Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
import base64
def lambda_handler(event, context):

    for record in event['Records']:
        try:
            print(f"Processed Kinesis Event - EventID: {record['eventID']}")
            record_data = base64.b64decode(record['kinesis']
['data']).decode('utf-8')
            print(f"Record Data: {record_data}")
            # TODO: Do interesting work based on the new data
        except Exception as e:
            print(f"An error occurred {e}")
            raise e
```

```
    print(f"Successfully processed {len(event['Records'])} records.")
```

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara Kinesis dengan Lambda menggunakan Ruby.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
require 'aws-sdk'

def lambda_handler(event:, context:)
  event['Records'].each do |record|
    begin
      puts "Processed Kinesis Event - EventID: #{record['eventID']}"
      record_data = get_record_data_async(record['kinesis'])
      puts "Record Data: #{record_data}"
      # TODO: Do interesting work based on the new data
    rescue => err
      $stderr.puts "An error occurred #{err}"
      raise err
    end
  end
  puts "Successfully processed #{event['Records'].length} records."
end

def get_record_data_async(payload)
  data = Base64.decode64(payload['data']).force_encoding('UTF-8')
  # Placeholder for actual async work
  # You can use Ruby's asynchronous programming tools like async/await or fibers
 here.
  return data
end
```

Rust

SDK for Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara Kinesis dengan Lambda menggunakan Rust.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::event::kinesis::KinesisEvent;
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

async fn function_handler(event: LambdaEvent<KinesisEvent>) -> Result<(), Error>
 {
    if event.payload.records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(());
    }

    event.payload.records.iter().for_each(|record| {
        tracing::info!("EventId:
 {}",record.event_id.as_deref().unwrap_or_default());

        let record_data = std::str::from_utf8(&record.kinesis.data);

        match record_data {
            Ok(data) => {
                // log the record data
                tracing::info!("Data: {}", data);
            }
            Err(e) => {
                tracing::error!("Error: {}", e);
            }
        }
    });

    tracing::info!(
        "Successfully processed {} records",
```

```
        event.payload.records.len()
    );

    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        // disable printing the name of the module in every log line.
        .with_target(false)
        // disabling time is handy because CloudWatch will add the ingestion
 time.
        .without_time()
        .init();

    run(service_fn(function_handler)).await
}
```

Untuk membuat Fungsi Acara

1. Buat direktori untuk proyek, dan kemudian beralih ke direktori itu.

```
mkdir kinesis-tutorial
cd kinesis-tutorial
```

2. Salin JavaScript kode sampel ke file baru bernama`index.js`.
3. Buat paket deployment.

```
zip function.zip index.js
```

4. Buat Fungsi Acara Lambda dengan perintah. `create-function`

```
aws lambda create-function --function-name ProcessKinesisRecords \
--zip-file fileb://function.zip --handler index.handler --runtime nodejs24.x \
--role arn:aws:iam::111122223333:role/lambda-kinesis-role
```

Uji Fungsi Peristiwa Lambda

Panggil Fungsi Peristiwa Lambda Anda secara manual menggunakan perintah `invoke` AWS Lambda CLI dan contoh peristiwa Kinesis.

Untuk menguji Fungsi Peristiwa Lambda

1. Salin JSON berikut ke dalam file dan simpan sebagai `input.txt`.

```
{
    "Records": [
        {
            "kinesis": {
                "kinesisSchemaVersion": "1.0",
                "partitionKey": "1",
                "sequenceNumber":
 "49590338271490256608559692538361571095921575989136588898",
                "data": "SGVsbG8sIHRoaXMgaXMgYSB0ZXN0Lg==",
                "approximateArrivalTimestamp": 1545084650.987
            },
            "eventSource": "aws:kinesis",
            "eventVersion": "1.0",
            "eventID":
 "shardId-000000000006:49590338271490256608559692538361571095921575989136588898",
            "eventName": "aws:kinesis:record",
            "invokeIdentityArn": "arn:aws:iam::111122223333:role/lambda-kinesis-
role",
            "awsRegion": "us-east-2",
            "eventSourceARN": "arn:aws:kinesis:us-east-2:111122223333:stream/
lambda-stream"
        }
    ]
}
```

2. Gunakan perintah `invoke` untuk mengirim kejadian ke fungsi.

```
aws lambda invoke --function-name ProcessKinesisRecords \
--cli-binary-format raw-in-base64-out \
--payload file://input.txt outputfile.txt
```

cli-binary-formatOpsi ini diperlukan jika Anda menggunakan AWS CLI versi 2. Untuk menjadikan ini pengaturan default, jalankan`aws configure set cli-binary-format raw-in-`

`base64-out`. Untuk informasi selengkapnya, lihat opsi baris perintah global yang AWS CLI didukung di Panduan AWS Command Line Interface Pengguna untuk Versi 2.

Respons disimpan ke `out.txt`.

Buat aliran Kinesis

Gunakan perintah `create-stream` untuk membuat aliran.

```
aws kinesis create-stream --stream-name lambda-stream --shard-count 1
```

Jalankan perintah `describe-stream` berikut untuk mendapatkan ARN aliran.

```
aws kinesis describe-stream --stream-name lambda-stream
```

Anda akan melihat output berikut:

```
{
    "StreamDescription": {
        "Shards": [
            {
                "ShardId": "shardId-000000000000",
                "HashKeyRange": {
                    "StartingHashKey": "0",
                    "EndingHashKey": "340282366920746074317682119384634633455"
                },
                "SequenceNumberRange": {
                    "StartingSequenceNumber":
 "49591073947768692513481539594623130411957558361251844610"
                }
            }
        ],
        "StreamARN": "arn:aws:kinesis:us-east-1:111122223333:stream/lambda-stream",
        "StreamName": "lambda-stream",
        "StreamStatus": "ACTIVE",
        "RetentionPeriodHours": 24,
        "EnhancedMonitoring": [
            {
                "ShardLevelMetrics": []
            }
        ],
```

```
        "EncryptionType": "NONE",
        "KeyId": null,
        "StreamCreationTimestamp": 1544828156.0
    }
}
```

Anda menggunakan ARN streaming di langkah berikutnya untuk mengaitkan aliran dengan Fungsi Acara Lambda Anda.

Tambahkan sumber acara di AWS Lambda

Jalankan perintah AWS CLI `add-event-source` berikut.

```
aws lambda create-event-source-mapping --function-name ProcessKinesisRecords \
--event-source  arn:aws:kinesis:us-east-1:111122223333:stream/lambda-stream \
--batch-size 100 --starting-position LATEST
```

Catat ID pemetaan untuk penggunaan selanjutnya. Anda bisa mendapatkan daftar pemetaan sumber kejadian dengan menjalankan perintah `list-event-source-mappings`.

```
aws lambda list-event-source-mappings --function-name ProcessKinesisRecords \
--event-source arn:aws:kinesis:us-east-1:111122223333:stream/lambda-stream
```

Dalam respons, Anda dapat memverifikasi nilai status adalah `enabled`. Pemetaan sumber peristiwa dapat dinonaktifkan untuk menjeda polling sementara tanpa kehilangan catatan.

Uji penyiapan

Untuk menguji pemetaan sumber kejadian, tambahkan rekaman kejadian ke aliran Kinesis Anda. `--data`Nilainya adalah string yang dikodekan CLI ke base64 sebelum mengirimnya ke Kinesis. Anda dapat menjalankan perintah yang sama lebih dari satu kali untuk menambahkan beberapa rekaman ke aliran.

```
aws kinesis put-record --stream-name lambda-stream --partition-key 1 \
--data "Hello, this is a test."
```

Lambda menggunakan peran eksekusi untuk membaca rekaman dari aliran. Kemudian ia memanggil Fungsi Peristiwa Lambda Anda, meneruskan batch catatan. Fungsi menerjemahkan data dari setiap catatan dan mencatatnya, mengirimkan output ke CloudWatch Log. Lihat log di konsol CloudWatch .

Bersihkan sumber daya Anda

Sekarang Anda dapat menghapus sumber daya yang Anda buat untuk tutorial ini, kecuali Anda ingin mempertahankannya. Dengan menghapus AWS sumber daya yang tidak lagi Anda gunakan, Anda mencegah tagihan yang tidak perlu ke Anda Akun AWS.

Untuk menghapus peran eksekusi

1. Buka halaman Peran dari konsol IAM.
2. Pilih peran eksekusi yang Anda buat.
3. Pilih Hapus.
4. Masukkan nama peran di bidang input teks dan pilih Hapus.

Untuk menghapus Fungsi Acara Lambda

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih Fungsi Acara yang Anda buat.
3. Pilih Tindakan, Hapus.
4. Ketik **`confirm`** kolom input teks dan pilih Hapus.

Untuk menghapus aliran Kinesis

1. Masuk ke Konsol Manajemen AWS dan buka konsol Kinesis di. https://console.aws.amazon.com/kinesis
2. Pilih aliran yang Anda buat.
3. Pilih Tindakan, Hapus.
4. Masukkan **`delete`** di bidang input teks.
5. Pilih Hapus.

Menggunakan Lambda dengan Kubernetes

Anda dapat menerapkan dan mengelola Lambda Event Functions dengan Kubernetes API AWS menggunakan Controllers for Kubernetes (ACK) atau Crossplane.

AWS Pengontrol untuk Kubernetes (ACK)

Anda dapat menggunakan ACK untuk menyebarkan dan mengelola AWS sumber daya dari Kubernetes API. Melalui ACK, AWS menyediakan pengontrol kustom open-source untuk AWS layanan seperti Lambda, Amazon Elastic Container Registry (Amazon ECR) Registry ECR), Amazon Simple Storage Service (Amazon S3), dan Amazon AI. SageMaker Setiap AWS layanan yang didukung memiliki pengontrol kustom sendiri. Di klaster Kubernetes Anda, instal pengontrol untuk setiap AWS layanan yang ingin Anda gunakan. Kemudian, buat Definisi Sumber Daya Kustom (CRD) untuk menentukan AWS sumber daya.

Kami menyarankan Anda menggunakan Helm 3.8 atau yang lebih baru untuk menginstal pengontrol ACK. Setiap pengontrol ACK dilengkapi dengan bagan Helm sendiri, yang menginstal controller, CRD, dan aturan RBAC Kubernetes. Untuk informasi selengkapnya, lihat Menginstal ACK Controller di dokumentasi ACK.

Setelah Anda membuat sumber daya kustom ACK, Anda dapat menggunakannya seperti objek Kubernetes bawaan lainnya. Misalnya, Anda dapat menerapkan dan mengelola Fungsi Peristiwa Lambda dengan toolchain Kubernetes pilihan Anda, termasuk kubectl.

Berikut adalah beberapa contoh kasus penggunaan untuk menyediakan Fungsi Peristiwa Lambda melalui ACK:

- Organisasi Anda menggunakan peran kontrol akses berbasis peran (RBAC) dan IAM untuk akun layanan guna membuat batas izin. Dengan ACK, Anda dapat menggunakan kembali model keamanan ini untuk Lambda tanpa harus membuat pengguna dan kebijakan baru.
- Organisasi Anda memiliki DevOps proses untuk menyebarkan sumber daya ke dalam klaster Amazon Elastic Kubernetes Service (Amazon EKS) menggunakan manifes Kubernetes. Dengan ACK, Anda dapat menggunakan manifes untuk menyediakan Lambda Event Functions tanpa membuat infrastruktur terpisah sebagai template kode.

Untuk informasi selengkapnya tentang penggunaan ACK, lihat tutorial Lambda di dokumentasi ACK.

Crossplane

Crossplane adalah proyek Cloud Native Computing Foundation (CNCF) open-source yang menggunakan Kubernetes untuk mengelola sumber daya infrastruktur cloud. Dengan Crossplane, pengembang dapat meminta infrastruktur tanpa perlu memahami kompleksitasnya. Tim platform mempertahankan kendali atas bagaimana infrastruktur disediakan dan dikelola.

Menggunakan Crossplane, Anda dapat menerapkan dan mengelola Fungsi Peristiwa Lambda dengan toolchain Kubernetes pilihan Anda seperti kubectl, dan pipeline apa pun yang dapat menyebarkan manifes ke Kubernetes. CI/CD Berikut adalah beberapa contoh kasus penggunaan untuk menyediakan Fungsi Peristiwa Lambda melalui Crossplane:

- Organisasi Anda ingin menegakkan kepatuhan dengan memastikan bahwa Fungsi Acara Lambda memiliki tag yang benar. Tim platform dapat menggunakan Komposisi Crossplane untuk menentukan kebijakan ini melalui abstraksi API. Pengembang kemudian dapat menggunakan abstraksi ini untuk menyebarkan Fungsi Peristiwa Lambda dengan tag.
- Proyek Anda menggunakan GitOps Kubernetes. Dalam model ini, Kubernetes terus-menerus merekonsiliasi repositori git (status yang diinginkan) dengan sumber daya yang berjalan di dalam cluster (status saat ini). Jika ada perbedaan, GitOps proses secara otomatis membuat perubahan pada cluster. Anda dapat menggunakan GitOps Kubernetes untuk menyebarkan dan mengelola Fungsi Peristiwa Lambda melalui Crossplane, menggunakan alat dan konsep Kubernetes yang sudah dikenal seperti CRD dan Controller.

Untuk mempelajari lebih lanjut tentang menggunakan Crossplane dengan Lambda, lihat berikut ini:

- AWS Blueprints for Crossplane: Repositori ini mencakup contoh cara menggunakan Crossplane untuk menyebarkan sumber daya, termasuk Lambda Event Functions. AWS

  > **Note**
  >
  > AWS Cetak biru untuk Crossplane sedang dalam pengembangan aktif dan tidak boleh digunakan dalam produksi.

- Menyebarkan Lambda dengan Amazon EKS dan Crossplane: Video ini menunjukkan contoh lanjutan dari penerapan AWS arsitektur tanpa server dengan Crossplane, mengeksplorasi desain dari perspektif pengembang dan platform.

Menggunakan Lambda dengan Amazon MQ

> **Note**
>
> Jika Anda ingin mengirim data ke target selain Fungsi Peristiwa Lambda atau memperkaya data sebelum mengirimnya, lihat Amazon Pipes. EventBridge

Amazon MQ adalah layanan broker pesan terkelola untuk Apache ActiveMQ dan RabbitMQ. Broker pesan memungkinkan aplikasi dan komponen perangkat lunak untuk berkomunikasi menggunakan berbagai bahasa pemrograman, sistem operasi, dan protokol pesan formal baik melalui topik atau tujuan acara antrian.

Amazon MQ juga dapat mengelola instans Amazon Elastic Compute Cloud (Amazon EC2) atas nama Anda dengan menginstal broker ActiveMQ atau RabbitMQ dan dengan menyediakan berbagai topologi jaringan dan kebutuhan infrastruktur lainnya.

Anda dapat menggunakan Fungsi Acara Lambda untuk memproses catatan dari broker pesan Amazon MQ Anda. Lambda memanggil fungsi Anda melalui pemetaan sumber peristiwa, sumber daya Lambda yang membaca pesan dari broker Anda dan memanggil fungsi secara serempak. Untuk opsi konfigurasi, lihatthe section called “Parameter”. Untuk kesalahan umum, lihatthe section called “Pemecahan Masalah”.

> **Warning**
>
> Pemetaan sumber peristiwa Lambda memproses setiap peristiwa setidaknya sekali, dan pemrosesan duplikat catatan dapat terjadi. Untuk menghindari potensi masalah yang terkait dengan duplikat peristiwa, kami sangat menyarankan agar Anda membuat kode fungsi Anda idempoten. Untuk mempelajari lebih lanjut, lihat Bagaimana cara membuat fungsi Lambda saya idempoten di Pusat Pengetahuan. AWS

Pemetaan sumber kejadian Amazon MQ memiliki pembatasan konfigurasi berikut:

- Konkurensi - Fungsi Peristiwa Lambda yang menggunakan pemetaan sumber peristiwa Amazon MQ memiliki pengaturan konkurensi maksimum default. Untuk ActiveMQ, layanan Lambda membatasi jumlah lingkungan eksekusi bersamaan hingga lima per pemetaan sumber peristiwa Amazon MQ. Untuk RabbitMQ, jumlah lingkungan eksekusi bersamaan dibatasi hingga 1 per pemetaan sumber peristiwa Amazon MQ.

Bahkan jika Anda mengubah setelan konkurensi yang dicadangkan atau disediakan fungsi, layanan Lambda tidak akan membuat lebih banyak lingkungan eksekusi tersedia. Untuk meminta peningkatan konkurensi maksimum default untuk satu pemetaan sumber peristiwa Amazon MQ, Dukungan hubungi UUID pemetaan sumber peristiwa, serta wilayah. Karena peningkatan diterapkan pada tingkat pemetaan sumber peristiwa tertentu, bukan tingkat akun atau wilayah, Anda perlu meminta peningkatan penskalaan secara manual untuk setiap pemetaan sumber peristiwa.

- Lintas akun — Lambda tidak mendukung pemrosesan lintas akun. Anda tidak dapat menggunakan Lambda untuk memproses catatan dari broker pesan Amazon MQ yang berbeda. Akun AWS
- Otentikasi - Untuk ActiveMQ, hanya ActiveMQ yang didukung. SimpleAuthenticationPlugin Untuk RabbitMQ, hanya mekanisme otentikasi PLAIN yang didukung. Pengguna harus menggunakan AWS Secrets Manager untuk mengelola kredensialnya. Untuk informasi selengkapnya tentang otentikasi ActiveMQ, lihat Mengintegrasikan broker ActiveMQ dengan LDAP di Panduan Pengembang Amazon MQ.
- Kuota koneksi – Broker memiliki jumlah maksimum koneksi yang diizinkan untuk setiap protokol wire-level. Kuota ini didasarkan pada jenis instans broker. Untuk informasi lebih lanjut, lihat bagian Broker dari Kuota di Amazon MQ dalam Panduan Developer Amazon MQ.
- Konektivitas – Anda dapat membuat broker di virtual private cloud (VPC) publik atau privat. Untuk VPC pribadi, Fungsi Acara Lambda Anda memerlukan akses ke VPC untuk menerima pesan. Untuk informasi lebih lanjut, lihat the section called “Konfigurasikan keamanan jaringan” nanti di bagian ini.
- Tujuan kejadian – Hanya tujuan antrean yang didukung. Namun, Anda dapat menggunakan topik virtual, yang berfungsi sebagai topik secara internal sambil berinteraksi dengan Lambda sebagai antrean. Untuk informasi lebih lanjut, lihat Tujuan Virtual di situs web Apache ActiveMQ, dan Host Virtual di situs web RabbitMQ.
- Topologi jaringan - Untuk ActiveMQ, hanya satu broker tunggal atau siaga yang didukung per pemetaan sumber peristiwa. Untuk RabbitMQ, hanya satu broker instans tunggal atau penerapan klaster yang didukung per pemetaan sumber peristiwa. Single-instance broker membutuhkan titik akhir failover. Untuk informasi selengkapnya tentang mode penyebaran broker ini, lihat Arsitektur Broker MQ Aktif dan Arsitektur Broker MQ Kelinci di Panduan Pengembang Amazon MQ.
- Protokol — Protokol yang didukung bergantung pada jenis integrasi Amazon MQ.
  - Untuk integrasi ActiveMQ, Lambda menggunakan pesan menggunakan OpenWire/Java protokol Message Service (JMS). Tidak ada protokol lain yang didukung untuk mengkonsumsi pesan. Dalam protokol JMS, hanya `TextMessage` dan `BytesMessage` yang didukung. Lambda juga

mendukung properti kustom JMS. Untuk informasi lebih lanjut tentang OpenWire protokol, lihat OpenWiredi situs web Apache ActiveMQ.

- Untuk integrasi RabbitMQ, Lambda mengkonsumsi pesan menggunakan protokol AMQP 0-9-1. Tidak ada protokol lain yang didukung untuk mengkonsumsi pesan. Untuk informasi lebih lanjut tentang implementasi protokol AMQP 0-9-1 RabbitMQ, lihat AMQP 0-9-1 Panduan Referensi Lengkap di situs web RabbitMQ.

Lambda secara otomatis mendukung versi terbaru ActiveMQ dan RabbitMQ yang didukung Amazon MQ. Untuk versi terbaru yang didukung, lihat catatan rilis Amazon MQ di Panduan Pengembang Amazon MQ.

> **Note**
>
> Secara default, Amazon MQ memiliki jangka waktu pemeliharaan mingguan untuk broker. Selama jangka waktu tersebut, broker tidak tersedia. Untuk broker tanpa standby, Lambda tidak dapat memproses pesan apa pun selama jangka waktu tersebut.

Topik

- Memahami grup konsumen Lambda untuk Amazon MQ
- Mengkonfigurasi sumber acara Amazon MQ untuk Lambda
- Parameter pemetaan sumber peristiwa
- Filter peristiwa dari sumber acara Amazon MQ
- Memecahkan masalah kesalahan pemetaan sumber peristiwa Amazon MQ

Memahami grup konsumen Lambda untuk Amazon MQ

Untuk berinteraksi dengan Amazon MQ, Lambda membuat grup konsumen yang dapat membaca dari broker Amazon MQ Anda. Grup konsumen dibuat dengan ID yang sama dengan UUID pemetaan sumber kejadian.

Untuk sumber peristiwa Amazon MQ, Lambda mengumpulkan catatan bersama dan mengirimkannya ke fungsi Anda dalam satu muatan. Untuk mengontrol perilaku, Anda dapat mengonfigurasi jendela batching dan ukuran batch. Lambda menarik pesan hingga memproses ukuran muatan maksimum 6 MB, jendela batching kedaluwarsa, atau jumlah catatan mencapai ukuran batch penuh. Untuk informasi selengkapnya, lihat Perilaku batching.

Grup konsumen mengambil pesan sebagai BLOB byte, base64-mengkodekannya menjadi satu muatan JSON, dan kemudian memanggil fungsi Anda. Jika fungsi Anda mengembalikan kesalahan untuk salah satu pesan dalam batch, Lambda mencoba ulang seluruh batch pesan sampai berhasil diproses atau pesan berakhir.

> **Note**
>
> Sementara Fungsi Acara Lambda biasanya memiliki batas waktu tunggu maksimum 15 menit, pemetaan sumber acara untuk Amazon MSK, Apache Kafka yang dikelola sendiri, Amazon DocumentDB, dan Amazon MQ untuk ActiveMQ dan RabbitMQ hanya mendukung fungsi dengan batas waktu tunggu maksimum 14 menit. Kendala ini memastikan bahwa pemetaan sumber peristiwa dapat menangani kesalahan fungsi dan percobaan ulang dengan benar.

Anda dapat memantau penggunaan konkurensi fungsi tertentu menggunakan `ConcurrentExecutions` metrik di Amazon CloudWatch. Untuk informasi selengkapnya tentang konkurensi, lihat the section called “Mengonfigurasi konkurensi terpesan”.

Example Acara rekam Amazon MQ

ActiveMQ

```
{
   "eventSource": "aws:mq",
   "eventSourceArn": "arn:aws:mq:us-
east-2:111122223333:broker:test:b-9bcfa592-423a-4942-879d-eb284b418fc8",
   "messages": [
      {
        "messageID": "ID:b-9bcfa592-423a-4942-879d-eb284b418fc8-1.mq.us-
east-2.amazonaws.com-37557-1234520418293-4:1:1:1:1",
        "messageType": "jms/text-message",
        "deliveryMode": 1,
        "replyTo": null,
        "type": null,
        "expiration": "60000",
        "priority": 1,
        "correlationId": "myJMSCoID",
        "redelivered": false,
        "destination": {
          "physicalName": "testQueue"
```

```
        },
        "data":"QUJDOkFBQUE=",
        "timestamp": 1598827811958,
        "brokerInTime": 1598827811958,
        "brokerOutTime": 1598827811959,
        "properties": {
          "index": "1",
          "doAlarm": "false",
          "myCustomProperty": "value"
        }
      },
      {
        "messageID": "ID:b-9bcfa592-423a-4942-879d-eb284b418fc8-1.mq.us-
east-2.amazonaws.com-37557-1234520418293-4:1:1:1:1",
        "messageType": "jms/bytes-message",
        "deliveryMode": 1,
        "replyTo": null,
        "type": null,
        "expiration": "60000",
        "priority": 2,
        "correlationId": "myJMSCoID1",
        "redelivered": false,
        "destination": {
          "physicalName": "testQueue"
        },
        "data":"LQaGQ82S48k=",
        "timestamp": 1598827811958,
        "brokerInTime": 1598827811958,
        "brokerOutTime": 1598827811959,
        "properties": {
          "index": "1",
          "doAlarm": "false",
          "myCustomProperty": "value"
        }
      }
   ]
}
```

RabbitMQ

```
{
  "eventSource": "aws:rmq",
```

```
  "eventSourceArn": "arn:aws:mq:us-
east-2:111122223333:broker:pizzaBroker:b-9bcfa592-423a-4942-879d-eb284b418fc8",
  "rmqMessagesByQueue": {
    "pizzaQueue::/": [
      {
        "basicProperties": {
          "contentType": "text/plain",
          "contentEncoding": null,
          "headers": {
            "header1": {
              "bytes": [
                118,
                97,
                108,
                117,
                101,
                49
              ]
            },
            "header2": {
              "bytes": [
                118,
                97,
                108,
                117,
                101,
                50
              ]
            },
            "numberInHeader": 10
          },
          "deliveryMode": 1,
          "priority": 34,
          "correlationId": null,
          "replyTo": null,
          "expiration": "60000",
          "messageId": null,
          "timestamp": "Jan 1, 1970, 12:33:41 AM",
          "type": null,
          "userId": "AIDACKCEVSQ6C2EXAMPLE",
          "appId": null,
          "clusterId": null,
          "bodySize": 80
        },
```

```
        "redelivered": false,
        "data": "eyJ0aW1lb3V0IjowLCJkYXRhIjoiQ1pybWYwR3c4T3Y0YnFMUXhENEUifQ=="
      }
    ]
  }
}
```

> **Note**
>
> Dalam contoh RabbitMQ, `pizzaQueue` adalah nama antrian RabbitMQ, dan `/` merupakan nama host virtual. Saat menerima pesan, sumber acara mencantumkan pesan di bawah`pizzaQueue::/`.

Mengkonfigurasi sumber acara Amazon MQ untuk Lambda

Topik

- Konfigurasikan keamanan jaringan
- Buat pemetaan sumber acara

Konfigurasikan keamanan jaringan

Untuk memberi Lambda akses penuh ke Amazon MQ melalui pemetaan sumber acara Anda, broker Anda harus menggunakan titik akhir publik (alamat IP publik), atau Anda harus memberikan akses ke VPC Amazon tempat Anda membuat broker.

Saat Anda menggunakan Amazon MQ dengan Lambda, buat titik akhir AWS PrivateLink VPC yang menyediakan akses Fungsi Acara ke sumber daya di VPC Amazon Anda.

> **Note**
>
> AWS PrivateLink Titik akhir VPC diperlukan untuk fungsi dengan pemetaan sumber peristiwa yang menggunakan mode default (sesuai permintaan) untuk poller acara. Jika pemetaan sumber acara menggunakan mode yang disediakan, Anda tidak perlu mengonfigurasi titik akhir AWS PrivateLink VPC.

Buat titik akhir untuk menyediakan akses ke sumber daya berikut:

- Lambda — Buat titik akhir untuk kepala layanan Lambda.
- AWS STS — Buat titik akhir AWS STS untuk prinsipal layanan untuk mengambil peran atas nama Anda.
- Secrets Manager — Jika broker Anda menggunakan Secrets Manager untuk menyimpan kredensi, buat endpoint untuk Secrets Manager.

Atau, konfigurasikan gateway NAT di setiap subnet publik di Amazon VPC. Untuk informasi selengkapnya, lihat the section called “Akses internet untuk fungsi VPC”.

Saat Anda membuat pemetaan sumber peristiwa untuk Amazon MQ, Lambda memeriksa apakah Antarmuka Jaringan Elastis (ENI) sudah ada untuk subnet dan grup keamanan yang dikonfigurasi untuk VPC Amazon Anda. Jika Lambda menemukan ENI yang ada, ia mencoba untuk menggunakannya kembali. Jika tidak, Lambda membuat ENI baru untuk terhubung ke sumber acara dan menjalankan fungsi Anda.

> **Note**
>
> Fungsi Acara Lambda selalu berjalan di dalam VPC yang dimiliki oleh layanan Lambda. Konfigurasi VPC fungsi Anda tidak memengaruhi pemetaan sumber peristiwa. Hanya konfigurasi jaringan sumber acara yang menentukan bagaimana Lambda terhubung ke sumber acara Anda.

Konfigurasikan grup keamanan untuk VPC Amazon yang berisi broker Anda. Secara default, Amazon MQ menggunakan port berikut: `61617` (Amazon MQ untuk ActiveMQ), `5671` dan (Amazon MQ untuk RabbitMQ).

- Aturan masuk - Izinkan semua lalu lintas di port broker default untuk grup keamanan yang terkait dengan sumber acara Anda. Atau, Anda dapat menggunakan aturan grup keamanan referensi mandiri untuk mengizinkan akses dari instans dalam grup keamanan yang sama.
- Aturan keluar — Izinkan semua lalu lintas di port `443` untuk tujuan eksternal jika fungsi Anda perlu berkomunikasi dengan AWS layanan. Atau, Anda juga dapat menggunakan aturan grup keamanan referensi diri untuk membatasi akses ke broker jika Anda tidak perlu berkomunikasi dengan layanan lain AWS .
- Aturan masuk titik akhir VPC Amazon — Jika Anda menggunakan titik akhir VPC Amazon, grup keamanan yang terkait dengan titik akhir VPC Amazon Anda harus mengizinkan lalu lintas masuk di port dari grup keamanan broker. `443`

Jika broker Anda menggunakan otentikasi, Anda juga dapat membatasi kebijakan titik akhir untuk titik akhir Secrets Manager. Untuk memanggil Secrets Manager API, Lambda menggunakan peran fungsi Anda, bukan kepala layanan Lambda.

Example Kebijakan titik akhir VPC — titik akhir Secrets Manager

```
{
      "Statement": [
          {
              "Action": "secretsmanager:GetSecretValue",
              "Effect": "Allow",
              "Principal": {
                  "AWS": [
                      "arn:aws::iam::123456789012:role/my-role"
                  ]
              },
              "Resource": "arn:aws::secretsmanager:us-west-2:123456789012:secret:my-
secret"
          }
      ]
  }
```

Saat Anda menggunakan titik akhir Amazon VPC, AWS merutekan panggilan API Anda untuk menjalankan fungsi Anda menggunakan Antarmuka Jaringan Elastis (ENI) titik akhir. Kepala layanan Lambda perlu memanggil `lambda:InvokeFunction` peran dan fungsi apa pun yang menggunakan ENI tersebut.

Secara default, titik akhir Amazon VPC memiliki kebijakan IAM terbuka yang memungkinkan akses luas ke sumber daya. Praktik terbaik adalah membatasi kebijakan ini untuk melakukan tindakan yang diperlukan menggunakan titik akhir tersebut. Untuk memastikan bahwa pemetaan sumber peristiwa Anda dapat menjalankan Fungsi Peristiwa Lambda Anda, kebijakan titik akhir VPC harus mengizinkan kepala layanan Lambda untuk memanggil dan. `sts:AssumeRole` `lambda:InvokeFunction` Membatasi kebijakan titik akhir VPC Anda agar hanya mengizinkan panggilan API yang berasal dari organisasi Anda mencegah pemetaan sumber peristiwa berfungsi dengan baik, `"Resource": "*"` sehingga diperlukan dalam kebijakan ini.

Contoh kebijakan titik akhir VPC berikut menunjukkan cara memberikan akses yang diperlukan ke prinsipal layanan Lambda untuk titik akhir dan Lambda. AWS STS

Example Kebijakan Titik Akhir VPC — AWS STS titik akhir

```
{
      "Statement": [
          {
              "Action": "sts:AssumeRole",
              "Effect": "Allow",
              "Principal": {
                  "Service": [
                      "lambda.amazonaws.com"
                  ]
              },
              "Resource": "*"
          }
      ]
    }
```

Example Kebijakan Titik Akhir VPC - Titik akhir Lambda

```
{
      "Statement": [
          {
              "Action": "lambda:InvokeFunction",
              "Effect": "Allow",
              "Principal": {
                  "Service": [
                      "lambda.amazonaws.com"
                  ]
              },
              "Resource": "*"
          }
      ]
  }
```

Buat pemetaan sumber acara

Buat pemetaan sumber acara untuk memberi tahu Lambda agar mengirim catatan dari broker Amazon MQ ke Fungsi Acara Lambda. Anda dapat membuat beberapa pemetaan sumber kejadian untuk memproses data yang sama dengan beberapa fungsi, atau untuk memproses item dari beberapa sumber dengan satu fungsi.

Untuk mengonfigurasi fungsi agar dibaca dari Amazon MQ, tambahkan izin yang diperlukan dan buat pemicu MQ di konsol Lambda.

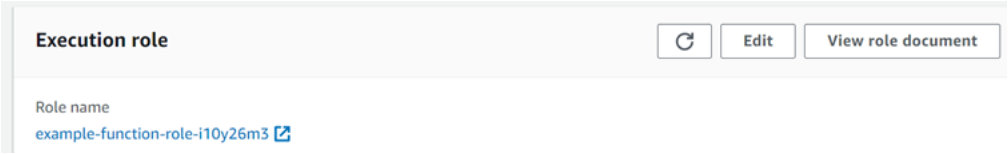
Untuk membaca catatan dari broker Amazon MQ, Fungsi Acara Lambda Anda memerlukan izin berikut. Anda memberi Lambda izin untuk berinteraksi dengan broker Amazon MQ Anda dan sumber yang mendasarinya dengan menambahkan pernyataan izin ke peran eksekusi fungsi Anda:

- mq: DescribeBroker
- manajer rahasia: GetSecretValue
- EC2: CreateNetworkInterface
- EC2: DeleteNetworkInterface
- EC2: DescribeNetworkInterfaces
- EC2: DescribeSecurityGroups
- EC2: DescribeSubnets
- EC2: DescribeVpcs
- log: CreateLogGroup
- log: CreateLogStream
- log: PutLogEvents

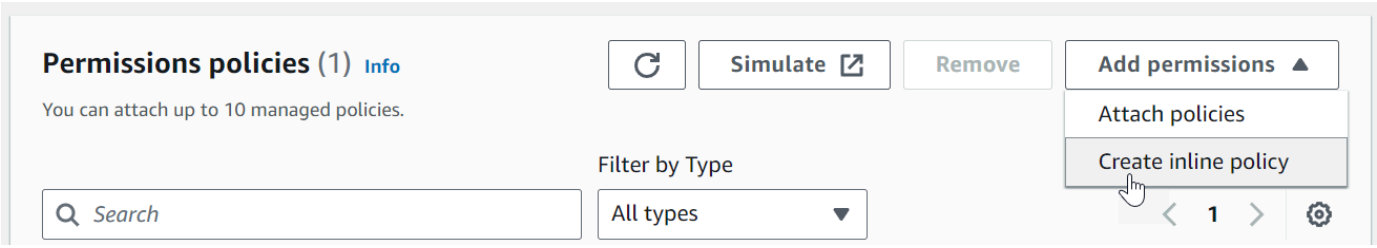
> **Note**
>
> Saat menggunakan kunci terkelola pelanggan yang terenkripsi, tambahkan juga izin `kms:Decrypt`.

Untuk menambahkan izin dan membuat pemicu

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih nama sebuah fungsi.
3. Pilih tab Konfigurasi, lalu pilih Izin.
4. Di bawah Nama peran, pilih tautan ke peran eksekusi Anda. Tautan ini membuka peran di konsol IAM.



5. Pilih Tambahkan izin, lalu pilih Buat kebijakan sebaris.



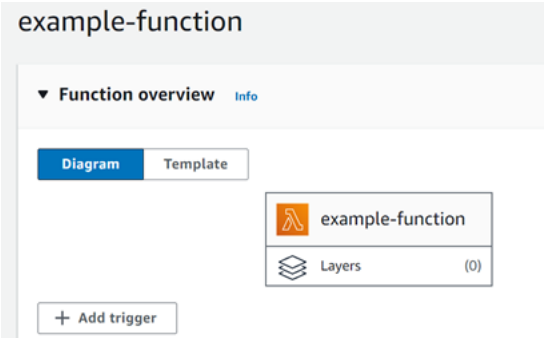
6. Di editor Kebijakan, pilih JSON. Masukkan kebijakan berikut. Fungsi Anda memerlukan izin ini untuk membaca dari broker Amazon MQ.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Action": [
          "mq:DescribeBroker",
          "secretsmanager:GetSecretValue",
          "ec2:CreateNetworkInterface",
          "ec2:DeleteNetworkInterface",
          "ec2:DescribeNetworkInterfaces",
          "ec2:DescribeSecurityGroups",
          "ec2:DescribeSubnets",
          "ec2:DescribeVpcs",
          "logs:CreateLogGroup",
          "logs:CreateLogStream",
          "logs:PutLogEvents"
        ],
        "Resource": "*"
      }
    ]
  }
```

> **Note**
>
> Saat menggunakan kunci terkelola pelanggan terenkripsi, Anda juga harus menambahkan izin. `kms:Decrypt`

7. Pilih Berikutnya. Masukkan nama kebijakan, lalu pilih Buat kebijakan.
8. Kembali ke fungsi Anda di konsol Lambda. Di bagian Gambaran umum fungsi, pilih Tambah pemicu.



9. Pilih jenis pemicu MQ.
10. Konfigurasikan opsi yang diperlukan, lalu pilih Tambah.

Lambda mendukung opsi berikut untuk sumber kejadian Amazon MQ:

- MQ broker – Pilih broker Amazon MQ.
- Ukuran batch – Atur jumlah maksimum pesan yang akan diambil dalam satu batch.
- Nama antrean – Masukkan antrean Amazon MQ yang akan digunakan.
- Konfigurasi akses sumber — Masukkan informasi host virtual dan rahasia Secrets Manager yang menyimpan kredensi broker Anda.
- Aktifkan pemicu – Nonaktifkan pemicu untuk menghentikan pemrosesan rekaman.

Untuk mengaktifkan atau menonaktifkan pemicu (atau menghapusnya) pilih pemicu MQ di desainer. Untuk mengonfigurasi ulang pemicu, gunakan operasi API pemetaan sumber kejadian.

Parameter pemetaan sumber peristiwa

Semua jenis sumber peristiwa Lambda berbagi operasi yang sama CreateEventSourceMappingdan UpdateEventSourceMappingAPI. Namun, hanya beberapa parameter yang berlaku untuk Amazon MQ dan RabbitMQ.

Parameter	Diperlukan	Default	Catatan
BatchSize	T	100	Maksimum: 10.000.
Diaktifkan	T	true	none
FunctionName	T	N/A	none
FilterCriteria	T	N/A	Kontrol peristiwa mana yang dikirim Lambda ke Fungsi Acara Anda
MaximumBatchingWindowInSeconds	T	500 ms	Perilaku batching
Antrean	T	N/A	Nama antrean tujuan broker Amazon MQ yang akan digunakan.
SourceAccessConfigurations	T	N/A	Untuk ActiveMQ, kredensial BASIC_AUTH. Untuk RabbitMQ, dapat berisi kredensial BASIC_AUTH dan informasi VIRTUAL_HOST.

Filter peristiwa dari sumber acara Amazon MQ

Anda dapat menggunakan pemfilteran peristiwa untuk mengontrol rekaman mana dari aliran atau antrian yang dikirim Lambda ke Fungsi Acara Anda. Untuk informasi umum tentang cara kerja penyaringan acara, lihatthe section called “Pemfilteran peristiwa”.

Bagian ini berfokus pada pemfilteran acara untuk sumber acara Amazon MQ.

> **Note**
>
> Pemetaan sumber acara Amazon MQ hanya mendukung pemfilteran pada kunci. `data`

Topik

- Dasar-dasar penyaringan acara Amazon MQ

Dasar-dasar penyaringan acara Amazon MQ

Misalkan antrian pesan Amazon MQ Anda berisi pesan baik dalam format JSON yang valid atau sebagai string biasa. Contoh catatan akan terlihat seperti berikut, dengan data dikonversi ke string yang dikodekan Base64 di lapangan. `data`

ActiveMQ

```
{
    "messageID": "ID:b-9bcfa592-423a-4942-879d-eb284b418fc8-1.mq.us-
east-2.amazonaws.com-37557-1234520418293-4:1:1:1:1",
    "messageType": "jms/text-message",
    "deliveryMode": 1,
    "replyTo": null,
    "type": null,
    "expiration": "60000",
    "priority": 1,
    "correlationId": "myJMSCoID",
    "redelivered": false,
    "destination": {
      "physicalName": "testQueue"
    },
    "data":"QUJDOkFBQUE=",
    "timestamp": 1598827811958,
    "brokerInTime": 1598827811958,
```

```
    "brokerOutTime": 1598827811959,
    "properties": {
      "index": "1",
      "doAlarm": "false",
      "myCustomProperty": "value"
    }
}
```

RabbitMQ

```
{
    "basicProperties": {
        "contentType": "text/plain",
        "contentEncoding": null,
        "headers": {
            "header1": {
                "bytes": [
                  118,
                  97,
                  108,
                  117,
                  101,
                  49
                ]
            },
            "header2": {
                "bytes": [
                  118,
                  97,
                  108,
                  117,
                  101,
                  50
                ]
            },
            "numberInHeader": 10
        },
        "deliveryMode": 1,
        "priority": 34,
        "correlationId": null,
        "replyTo": null,
        "expiration": "60000",
        "messageId": null,
```

```
        "timestamp": "Jan 1, 1970, 12:33:41 AM",
        "type": null,
        "userId": "AIDACKCEVSQ6C2EXAMPLE",
        "appId": null,
        "clusterId": null,
        "bodySize": 80
        },
    "redelivered": false,
    "data": "eyJ0aW1lb3V0IjowLCJkYXRhIjoiQ1pybWYwR3c4T3Y0YnFMUXhENEUifQ=="
}
```

Untuk broker MQ Aktif dan MQ Kelinci, Anda dapat menggunakan penyaringan acara untuk memfilter catatan menggunakan kunci. `data` Misalkan antrian Amazon MQ Anda berisi pesan dalam format JSON berikut.

```
{
    "timeout": 0,
    "IPAddress": "203.0.113.254"
}
```

Untuk memfilter hanya catatan yang `timeout` bidangnya lebih besar dari 0, `FilterCriteria` objeknya adalah sebagai berikut.

```
{
    "Filters": [
        {
            "Pattern": "{ \"data\" : { \"timeout\" : [ { \"numeric\": [ \">\",
 0] } } ] } }"
        }
    ]
}
```

Untuk kejelasan tambahan, berikut adalah nilai filter yang `Pattern` diperluas di JSON biasa.

```
{
    "data": {
        "timeout": [ { "numeric": [ ">", 0 ] } ]
        }
}
```

Anda dapat menambahkan filter menggunakan konsol, AWS CLI atau AWS SAM templat.

Console

untuk menambahkan filter ini menggunakan konsol, ikuti instruksi Melampirkan kriteria filter ke pemetaan sumber peristiwa (konsol) dan masukkan string berikut untuk kriteria Filter.

```
{ "data" : { "timeout" : [ { "numeric": [ ">", 0 ] } ] } }
```

AWS CLI

Untuk membuat pemetaan sumber peristiwa baru dengan kriteria filter ini menggunakan AWS Command Line Interface (AWS CLI), jalankan perintah berikut.

```
aws lambda create-event-source-mapping \
    --function-name my-function \
    --event-source-arn arn:aws:mq:us-east-2:123456789012:broker:my-
broker:b-8ac7cc01-5898-482d-be2f-a6b596050ea8 \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"data\" : { \"timeout\" :
 [ { \"numeric\": [ \">\", 0 ] } ] } }"}]}'
```

Untuk menambahkan kriteria filter ini ke pemetaan sumber peristiwa yang ada, jalankan perintah berikut.

```
aws lambda update-event-source-mapping \
    --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"data\" : { \"timeout\" :
 [ { \"numeric\": [ \">\", 0 ] } ] } }"}]}'
```

Untuk menambahkan kriteria filter ini ke pemetaan sumber peristiwa yang ada, jalankan perintah berikut.

```
aws lambda update-event-source-mapping \
    --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"data\" : { \"timeout\" :
 [ { \"numeric\": [ \">\", 0 ] } ] } }"}]}'
```

AWS SAM

Untuk menambahkan filter ini menggunakan AWS SAM, tambahkan cuplikan berikut ke template YAMAL untuk sumber acara Anda.

```
FilterCriteria:
  Filters:
    - Pattern: '{ "data" : { "timeout" : [ { "numeric": [ ">", 0 ] } ] } }'
```

Dengan Amazon MQ, Anda juga dapat memfilter catatan di mana pesan adalah string biasa. Misalkan Anda hanya ingin memproses catatan di mana pesan dimulai dengan “Hasil:”. `FilterCriteria`Objek akan terlihat sebagai berikut.

```
{
    "Filters": [
        {
            "Pattern": "{ \"data\" : [ { \"prefix\": \"Result: \" } ] }"
        }
    ]
}
```

Untuk kejelasan tambahan, berikut adalah nilai filter yang `Pattern` diperluas di JSON biasa.

```
{
    "data": [
        {
        "prefix": "Result: "
        }
    ]
}
```

Anda dapat menambahkan filter menggunakan konsol, AWS CLI atau AWS SAM templat.

Console

Untuk menambahkan filter ini menggunakan konsol, ikuti instruksi Melampirkan kriteria filter ke pemetaan sumber peristiwa (konsol) dan masukkan string berikut untuk kriteria Filter.

```
{ "data" : [ { "prefix": "Result: " } ] }
```

AWS CLI

Untuk membuat pemetaan sumber peristiwa baru dengan kriteria filter ini menggunakan AWS Command Line Interface (AWS CLI), jalankan perintah berikut.

```
aws lambda create-event-source-mapping \
    --function-name my-function \
    --event-source-arn arn:aws:mq:us-east-2:123456789012:broker:my-
broker:b-8ac7cc01-5898-482d-be2f-a6b596050ea8 \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"data\" : [ { \"prefix\":
 \"Result: \" } ] }"}]}'
```

Untuk menambahkan kriteria filter ini ke pemetaan sumber peristiwa yang ada, jalankan perintah berikut.

```
aws lambda update-event-source-mapping \
    --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"data\" : [ { \"prefix\":
 \"Result: \" } ] }"}]}'
```

AWS SAM

Untuk menambahkan filter ini menggunakan AWS SAM, tambahkan cuplikan berikut ke template YAMAL untuk sumber acara Anda.

```
FilterCriteria:
  Filters:
    - Pattern: '{ "data" : [ { "prefix": "Result " } ] }'
```

Pesan Amazon MQ harus berupa string yang UTF-8 dikodekan, baik string biasa atau dalam format JSON. Itu karena Lambda menerjemahkan array byte Amazon MQ ke dalam sebelum menerapkan kriteria filter. UTF-8 Jika pesan Anda menggunakan pengkodean lain, seperti UTF-16 atau ASCII, atau jika format pesan tidak cocok dengan format`FilterCriteria`, Lambda hanya memproses filter metadata. Tabel berikut merangkum perilaku spesifik:

Format pesan masuk	Format pola filter untuk properti pesan	Tindakan yang dihasilkan
Tali polos	Tali polos	Filter Lambda berdasarkan kriteria filter Anda.
Tali polos	Tidak ada pola filter untuk properti data	Filter Lambda (hanya pada properti metadata lainnya)

Format pesan masuk	Format pola filter untuk properti pesan	Tindakan yang dihasilkan
		berdasarkan kriteria filter Anda.
Tali polos	JSON yang valid	Filter Lambda (hanya pada properti metadata lainnya) berdasarkan kriteria filter Anda.
JSON yang valid	Tali polos	Filter Lambda (hanya pada properti metadata lainnya) berdasarkan kriteria filter Anda.
JSON yang valid	Tidak ada pola filter untuk properti data	Filter Lambda (hanya pada properti metadata lainnya) berdasarkan kriteria filter Anda.
JSON yang valid	JSON yang valid	Filter Lambda berdasarkan kriteria filter Anda.
Non-UTF-8 string yang dikodekan	JSON, string polos, atau tidak ada pola	Filter Lambda (hanya pada properti metadata lainnya) berdasarkan kriteria filter Anda.

Memecahkan masalah kesalahan pemetaan sumber peristiwa Amazon MQ

Saat Fungsi Peristiwa Lambda mengalami kesalahan yang tidak dapat dipulihkan, konsumen Amazon MQ Anda berhenti memproses catatan. Konsumen lain dapat melanjutkan pemrosesan, asalkan mereka tidak mengalami kesalahan yang sama. Untuk menentukan kemungkinan penyebab dari konsumen yang dihentikan, periksa bidang `StateTransitionReason` dalam detail pengembalian `EventSourceMapping` untuk salah satu kode berikut:

`ESM_CONFIG_NOT_VALID`

Konfigurasi pemetaan sumber kejadian tidak valid.

`EVENT_SOURCE_AUTHN_ERROR`

Lambda gagal mengautentikasi sumber kejadian.

`EVENT_SOURCE_AUTHZ_ERROR`

Lambda tidak memiliki izin yang diperlukan untuk mengakses sumber kejadian.

`FUNCTION_CONFIG_NOT_VALID`

Konfigurasi fungsi tidak valid.

Catatan juga tidak diproses jika Lambda menjatuhkannya karena ukurannya. Batas ukuran untuk rekaman Lambda adalah 6 MB. Untuk mengirim ulang pesan setelah kesalahan fungsi, Anda dapat menggunakan antrian huruf mati (DLQ). Untuk informasi selengkapnya, lihat Pengiriman Ulang Pesan dan Penanganan DLQ di situs web Apache ActiveMQ dan Panduan Keandalan di situs web RabbitMQ.

> **Note**
>
> Lambda tidak mendukung kebijakan pengiriman ulang khusus. Sebagai gantinya, Lambda menggunakan kebijakan dengan nilai default dari halaman Kebijakan Pengiriman Ulang di situs web Apache ActiveMQ, dengan disetel ke 6. `maximumRedeliveries`

Penggunaan AWS Lambda dengan Amazon RDS

Anda dapat menghubungkan Fungsi Peristiwa Lambda ke database Amazon Relational Database Service (Amazon RDS) secara langsung dan melalui Amazon RDS Proxy. Koneksi langsung berguna dalam skenario sederhana, dan proxy direkomendasikan untuk produksi. Proksi basis data mengelola kumpulan koneksi basis data bersama yang memungkinkan fungsi Anda mencapai tingkat konkurensi tinggi tanpa menghabiskan koneksi basis data.

Untuk membantu menentukan apakah akan menggunakan Amazon RDS atau DynamoDB untuk beban kerja Anda, lihat. the section called “Amazon RDS vs DynamoDB”

Sebaiknya gunakan Amazon RDS Proxy untuk Lambda Event Functions yang sering membuat koneksi database pendek, atau membuka dan menutup sejumlah besar koneksi database. Untuk informasi selengkapnya, lihat Menghubungkan fungsi Lambda dan instans DB secara otomatis di Panduan Pengembang Layanan Amazon Relational Database Service.

> **Tip**
>
> Untuk menghubungkan Fungsi Peristiwa Lambda dengan cepat ke database Amazon RDS, Anda dapat menggunakan panduan panduan dalam konsol. Untuk membuka wizard, lakukan hal berikut:
>
> 1. Buka halaman Fungsi Acara konsol Lambda.
> 2. Pilih fungsi yang ingin Anda sambungkan ke database.
> 3. Pada tab Konfigurasi, pilih database RDS.
> 4. Pilih Connect to RDS database.
>
> Setelah menghubungkan fungsi ke database, Anda dapat membuat proxy dengan memilih Add proxy.

Mengkonfigurasi Fungsi Acara Anda untuk bekerja dengan sumber daya RDS

Di konsol Lambda, Anda dapat menyediakan, dan mengonfigurasi, instans database Amazon RDS dan sumber daya proxy. Anda dapat melakukan ini dengan menavigasi ke database RDS di bawah tab Konfigurasi. Atau, Anda juga dapat membuat dan mengonfigurasi koneksi ke Lambda Event Functions di konsol Amazon RDS. Saat mengonfigurasi instance database RDS untuk digunakan dengan Lambda, perhatikan kriteria berikut:

- Untuk terhubung ke database, fungsi Anda harus berada di VPC Amazon yang sama tempat database Anda berjalan.
- Anda dapat menggunakan database Amazon RDS dengan mesin MySQL, MariaDB, PostgreSQL, atau Microsoft SQL Server.
- Anda juga dapat menggunakan cluster Aurora DB dengan mesin MySQL atau PostgreSQL.
- Anda perlu memberikan rahasia Secrets Manager untuk otentikasi database.
- Peran IAM harus memberikan izin untuk menggunakan rahasia, dan kebijakan kepercayaan harus memungkinkan Amazon RDS untuk mengambil peran tersebut.
- Prinsipal IAM yang menggunakan konsol untuk mengonfigurasi sumber daya Amazon RDS, dan menghubungkannya ke fungsi Anda harus memiliki izin berikut:

Contoh kebijakan izin

> **Note**
>
> Anda memerlukan izin Amazon RDS Proxy hanya jika Anda mengonfigurasi Proxy Amazon RDS untuk mengelola kumpulan koneksi database Anda.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:CreateSecurityGroup",
        "ec2:DescribeSecurityGroups",
        "ec2:DescribeSubnets",
        "ec2:DescribeVpcs",
        "ec2:AuthorizeSecurityGroupIngress",
        "ec2:AuthorizeSecurityGroupEgress",
        "ec2:RevokeSecurityGroupEgress",
        "ec2:CreateNetworkInterface",
        "ec2:DeleteNetworkInterface",
        "ec2:DescribeNetworkInterfaces"
      ],
```

```
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "rds-db:connect",
        "rds:CreateDBProxy",
        "rds:CreateDBInstance",
        "rds:CreateDBSubnetGroup",
        "rds:DescribeDBClusters",
        "rds:DescribeDBInstances",
        "rds:DescribeDBSubnetGroups",
        "rds:DescribeDBProxies",
        "rds:DescribeDBProxyTargets",
        "rds:DescribeDBProxyTargetGroups",
        "rds:RegisterDBProxyTargets",
        "rds:ModifyDBInstance",
        "rds:ModifyDBProxy"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "lambda:CreateFunction",
        "lambda:ListFunctions",
        "lambda:UpdateFunctionConfiguration"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "iam:AttachRolePolicy",
        "iam:CreateRole",
        "iam:CreatePolicy"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "secretsmanager:GetResourcePolicy",
        "secretsmanager:GetSecretValue",
```

```
        "secretsmanager:DescribeSecret",
        "secretsmanager:ListSecretVersionIds",
        "secretsmanager:CreateSecret"
      ],
      "Resource": "*"
    }
  ]
}
```

Amazon RDS mengenakan tarif per jam untuk proxy berdasarkan ukuran instans database, lihat harga Proxy RDS untuk detailnya. Untuk informasi selengkapnya tentang koneksi proxy secara umum, lihat Menggunakan Proxy Amazon RDS di Panduan Pengguna Amazon RDS.

SSL/TLS persyaratan untuk koneksi Amazon RDS

Untuk membuat SSL/TLS sambungan aman ke instans database Amazon RDS, Fungsi Peristiwa Lambda Anda harus memverifikasi identitas server database menggunakan sertifikat tepercaya. Lambda menangani sertifikat ini secara berbeda tergantung pada jenis paket penerapan Anda:

- arsip file.zip: Penanganan sertifikat bervariasi menurut runtime:
  - Node.js 18 dan sebelumnya: Lambda secara otomatis menyertakan sertifikat CA dan sertifikat RDS.
  - Node.js 20 dan yang lebih baru: Lambda tidak lagi memuat sertifikat CA tambahan secara default. Atur `NODE_EXTRA_CA_CERTS` variabel lingkungan ke `/var/runtime/ca-cert.pem`.

  Mungkin diperlukan waktu hingga 4 minggu untuk sertifikat Amazon RDS Wilayah AWS agar baru ditambahkan ke runtime terkelola Lambda.
- Gambar kontainer: gambar AWS dasar hanya menyertakan sertifikat CA. Jika fungsi Anda tersambung ke instans database Amazon RDS, Anda harus menyertakan sertifikat yang sesuai dalam image container Anda. Di Dockerfile Anda, unduh bundel sertifikat yang sesuai dengan Wilayah AWS tempat Anda meng-host database Anda. Contoh:

  ```
  RUN curl https://truststore.pki.rds.amazonaws.com/us-east-1/us-east-1-bundle.pem -o /us-east-1-bundle.pem
  ```

Perintah ini mengunduh bundel sertifikat Amazon RDS dan menyimpannya di jalur absolut `/us-east-1-bundle.pem` di direktori root container Anda. Saat mengonfigurasi koneksi database dalam kode fungsi Anda, Anda harus mereferensikan jalur yang tepat ini. Contoh:

Node.js

`readFileSync`Fungsi ini diperlukan karena klien Node.js database memerlukan konten sertifikat yang sebenarnya dalam memori, bukan hanya jalur ke file sertifikat. Tanpa`readFileSync`, klien menafsirkan string jalur sebagai konten sertifikat, menghasilkan kesalahan "sertifikat yang ditandatangani sendiri dalam rantai sertifikat".

Example Node.js konfigurasi koneksi untuk fungsi OCI

```
import { readFileSync } from 'fs';

// ...

let connectionConfig = {
    host: process.env.ProxyHostName,
    user: process.env.DBUserName,
    password: token,
    database: process.env.DBName,
    ssl: {
        ca: readFileSync('/us-east-1-bundle.pem') // Load RDS certificate content
 from file into memory
    }
};
```

Python

Example Konfigurasi koneksi Python untuk fungsi OCI

```
connection = pymysql.connect(
    host=proxy_host_name,
    user=db_username,
    password=token,
    db=db_name,
    port=port,
    ssl={'ca': '/us-east-1-bundle.pem'}  #Path to the certificate in container
)
```

Java

Untuk fungsi Java yang menggunakan koneksi JDBC, string koneksi harus mencakup:

- `useSSL=true`

- `requireSSL=true`
- `sslCA`Parameter yang menunjuk ke lokasi sertifikat Amazon RDS dalam gambar kontainer

Example String koneksi Java untuk fungsi OCI

```
// Define connection string
String connectionString = String.format("jdbc:mysql://%s:%s/%s?
useSSL=true&requireSSL=true&sslCA=/us-east-1-bundle.pem", // Path to the certificate
 in container
        System.getenv("ProxyHostName"),
        System.getenv("Port"),
        System.getenv("DBName"));
```

.NET

Example.NET string koneksi untuk koneksi MySQL dalam fungsi OCI

```
/// Build the Connection String with the Token
string connectionString =
 $"Server={Environment.GetEnvironmentVariable("RDS_ENDPOINT")};" +
                         $"Port={Environment.GetEnvironmentVariable("RDS_PORT")};" +

 $"Uid={Environment.GetEnvironmentVariable("RDS_USERNAME")};" +
                         $"Pwd={authToken};" +
                         "SslMode=Required;" +
                         "SslCa=/us-east-1-bundle.pem";  // Path to the certificate
 in container
```

Go

Untuk fungsi Go yang menggunakan koneksi MySQL, muat sertifikat Amazon RDS ke dalam kumpulan sertifikat dan daftarkan dengan driver MySQL. String koneksi kemudian harus mereferensikan konfigurasi ini menggunakan `tls` parameter.

Example Kode Go untuk koneksi MySQL dalam fungsi OCI

```
import (
    "crypto/tls"
    "crypto/x509"
    "os"
    "github.com/go-sql-driver/mysql"
```

```
)

...

// Create certificate pool and register TLS config
rootCertPool := x509.NewCertPool()
pem, err := os.ReadFile("/us-east-1-bundle.pem")  // Path to the certificate in
 container
if err != nil {
    panic("failed to read certificate file: " + err.Error())
}
if ok := rootCertPool.AppendCertsFromPEM(pem); !ok {
    panic("failed to append PEM")
}

mysql.RegisterTLSConfig("custom", &tls.Config{
    RootCAs: rootCertPool,
})

dsn := fmt.Sprintf("%s:%s@tcp(%s)/%s?allowCleartextPasswords=true&tls=custom",
    dbUser, authenticationToken, dbEndpoint, dbName,
)
```

Ruby

Example Konfigurasi koneksi Ruby untuk fungsi OCI

```
conn = Mysql2::Client.new(
    host: endpoint,
    username: user,
    password: token,
    port: port,
    database: db_name,
    sslca: '/us-east-1-bundle.pem',  # Path to the certificate in container
    sslverify: true
)
```

Menghubungkan ke database Amazon RDS dalam Fungsi Peristiwa Lambda

Contoh kode berikut menunjukkan cara mengimplementasikan Fungsi Peristiwa Lambda yang terhubung ke database Amazon RDS. Fungsi membuat permintaan database sederhana dan mengembalikan hasilnya.

> **Note**
>
> Contoh kode ini hanya berlaku untuk paket.zip deployment. Jika Anda menerapkan fungsi menggunakan gambar kontainer, Anda harus menentukan file sertifikat Amazon RDS dalam kode fungsi Anda, seperti yang dijelaskan di bagian sebelumnya.

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Menghubungkan ke database Amazon RDS dalam fungsi Lambda menggunakan.NET.

```
using System.Data;
using System.Text.Json;
using Amazon.Lambda.APIGatewayEvents;
using Amazon.Lambda.Core;
using MySql.Data.MySqlClient;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri

namespace aws_rds;

public class InputModel
{
    public string key1 { get; set; }
    public string key2 { get; set; }
}

public class Function
{
    /// <summary>
```

```
    // Handles the Lambda function execution for connecting to RDS using IAM
 authentication.
    /// </summary>
    /// <param name="input">The input event data passed to the Lambda function</
param>
    /// <param name="context">The Lambda execution context that provides runtime
 information</param>
    /// <returns>A response object containing the execution result</returns>

    public async Task<APIGatewayProxyResponse>
 FunctionHandler(APIGatewayProxyRequest request, ILambdaContext context)
    {
        // Sample Input: {"body": "{\"key1\":\"20\", \"key2\":\"25\"}"}
        var input = JsonSerializer.Deserialize<InputModel>(request.Body);

        /// Obtain authentication token
        var authToken = RDSAuthTokenGenerator.GenerateAuthToken(
            Environment.GetEnvironmentVariable("RDS_ENDPOINT"),
            Convert.ToInt32(Environment.GetEnvironmentVariable("RDS_PORT")),
            Environment.GetEnvironmentVariable("RDS_USERNAME")
        );

        /// Build the Connection String with the Token
        string connectionString =
 $"Server={Environment.GetEnvironmentVariable("RDS_ENDPOINT")};" +

 $"Port={Environment.GetEnvironmentVariable("RDS_PORT")};" +

 $"Uid={Environment.GetEnvironmentVariable("RDS_USERNAME")};" +
                                  $"Pwd={authToken};";


        try
        {
            await using var connection = new MySqlConnection(connectionString);
            await connection.OpenAsync();

            const string sql = "SELECT @param1 + @param2 AS Sum";

            await using var command = new MySqlCommand(sql, connection);
            command.Parameters.AddWithValue("@param1", int.Parse(input.key1 ??
 "0"));
            command.Parameters.AddWithValue("@param2", int.Parse(input.key2 ??
 "0"));
```

```
            await using var reader = await command.ExecuteReaderAsync();
            if (await reader.ReadAsync())
            {
                int result = reader.GetInt32("Sum");

                //Sample Response: {"statusCode":200,"body":"{\"message\":\"The
 sum is: 45\"}","isBase64Encoded":false}
                return new APIGatewayProxyResponse
                {
                    StatusCode = 200,
                    Body = JsonSerializer.Serialize(new { message = $"The sum is:
 {result}" })
                };
            }

        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error: {ex.Message}");
        }

        return new APIGatewayProxyResponse
        {
            StatusCode = 500,
            Body = JsonSerializer.Serialize(new { error = "Internal server
 error" })
        };
    }
}
```

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Menghubungkan ke database Amazon RDS dalam fungsi Lambda menggunakan Go.

```
/*
Golang v2 code here.
*/

package main

import (
 "context"
 "database/sql"
 "encoding/json"
 "fmt"
 "os"

 "github.com/aws/aws-lambda-go/lambda"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/feature/rds/auth"
 _ "github.com/go-sql-driver/mysql"
)

type MyEvent struct {
 Name string `json:"name"`
}

func HandleRequest(event *MyEvent) (map[string]interface{}, error) {

 var dbName string = os.Getenv("DatabaseName")
 var dbUser string = os.Getenv("DatabaseUser")
 var dbHost string = os.Getenv("DBHost") // Add hostname without https
 var dbPort int = os.Getenv("Port")      // Add port number
 var dbEndpoint string = fmt.Sprintf("%s:%d", dbHost, dbPort)
 var region string = os.Getenv("AWS_REGION")

 cfg, err := config.LoadDefaultConfig(context.TODO())
 if err != nil {
  panic("configuration error: " + err.Error())
 }

 authenticationToken, err := auth.BuildAuthToken(
  context.TODO(), dbEndpoint, region, dbUser, cfg.Credentials)
 if err != nil {
  panic("failed to create authentication token: " + err.Error())
 }
```

```
 dsn := fmt.Sprintf("%s:%s@tcp(%s)/%s?tls=true&allowCleartextPasswords=true",
  dbUser, authenticationToken, dbEndpoint, dbName,
 )

 db, err := sql.Open("mysql", dsn)
 if err != nil {
  panic(err)
 }

 defer db.Close()

 var sum int
 err = db.QueryRow("SELECT ?+? AS sum", 3, 2).Scan(&sum)
 if err != nil {
  panic(err)
 }
 s := fmt.Sprint(sum)
 message := fmt.Sprintf("The selected sum is: %s", s)

 messageBytes, err := json.Marshal(message)
 if err != nil {
  return nil, err
 }

 messageString := string(messageBytes)
 return map[string]interface{}{
  "statusCode": 200,
  "headers":    map[string]string{"Content-Type": "application/json"},
  "body":       messageString,
 }, nil
}

func main() {
 lambda.Start(HandleRequest)
}
```

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Menghubungkan ke database Amazon RDS dalam fungsi Lambda menggunakan Java.

```
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.APIGatewayProxyRequestEvent;
import com.amazonaws.services.lambda.runtime.events.APIGatewayProxyResponseEvent;
import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rdsdata.RdsDataClient;
import software.amazon.awssdk.services.rdsdata.model.ExecuteStatementRequest;
import software.amazon.awssdk.services.rdsdata.model.ExecuteStatementResponse;
import software.amazon.awssdk.services.rdsdata.model.Field;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;

public class RdsLambdaHandler implements
 RequestHandler<APIGatewayProxyRequestEvent, APIGatewayProxyResponseEvent> {

    @Override
    public APIGatewayProxyResponseEvent handleRequest(APIGatewayProxyRequestEvent
 event, Context context) {
        APIGatewayProxyResponseEvent response = new
 APIGatewayProxyResponseEvent();

        try {
            // Obtain auth token
            String token = createAuthToken();

            // Define connection configuration
```

```
            String connectionString = String.format("jdbc:mysql://%s:%s/%s?
useSSL=true&requireSSL=true",
                    System.getenv("ProxyHostName"),
                    System.getenv("Port"),
                    System.getenv("DBName"));

            // Establish a connection to the database
            try (Connection connection =
 DriverManager.getConnection(connectionString, System.getenv("DBUserName"),
 token);
                 PreparedStatement statement =
 connection.prepareStatement("SELECT ? + ? AS sum")) {

                statement.setInt(1, 3);
                statement.setInt(2, 2);

                try (ResultSet resultSet = statement.executeQuery()) {
                    if (resultSet.next()) {
                        int sum = resultSet.getInt("sum");
                        response.setStatusCode(200);
                        response.setBody("The selected sum is: " + sum);
                    }
                }
            }

        } catch (Exception e) {
            response.setStatusCode(500);
            response.setBody("Error: " + e.getMessage());
        }

        return response;
    }

    private String createAuthToken() {
        // Create RDS Data Service client
        RdsDataClient rdsDataClient = RdsDataClient.builder()
                .region(Region.of(System.getenv("AWS_REGION")))
                .credentialsProvider(DefaultCredentialsProvider.create())
                .build();

        // Define authentication request
        ExecuteStatementRequest request = ExecuteStatementRequest.builder()
                .resourceArn(System.getenv("ProxyHostName"))
                .secretArn(System.getenv("DBUserName"))
```

```
                .database(System.getenv("DBName"))
                .sql("SELECT 'RDS IAM Authentication'")
                .build();

        // Execute request and obtain authentication token
        ExecuteStatementResponse response =
 rdsDataClient.executeStatement(request);
        Field tokenField = response.records().get(0).get(0);

        return tokenField.stringValue();
    }
}
```

JavaScript

SDK untuk JavaScript (v3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Menghubungkan ke database Amazon RDS dalam fungsi Lambda menggunakan. JavaScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
/*
Node.js code here.
*/
// ES6+ example
import { Signer } from "@aws-sdk/rds-signer";
import mysql from 'mysql2/promise';

async function createAuthToken() {
  // Define connection authentication parameters
  const dbinfo = {

    hostname: process.env.ProxyHostName,
    port: process.env.Port,
    username: process.env.DBUserName,
```

```
    region: process.env.AWS_REGION,

  }

  // Create RDS Signer object
  const signer = new Signer(dbinfo);

  // Request authorization token from RDS, specifying the username
  const token = await signer.getAuthToken();
  return token;
}

async function dbOps() {

  // Obtain auth token
  const token = await createAuthToken();
  // Define connection configuration
  let connectionConfig = {
    host: process.env.ProxyHostName,
    user: process.env.DBUserName,
    password: token,
    database: process.env.DBName,
    ssl: 'Amazon RDS'
  }
  // Create the connection to the DB
  const conn = await mysql.createConnection(connectionConfig);
  // Obtain the result of the query
  const [res,] = await conn.execute('select ?+? as sum', [3, 2]);
  return res;

}

export const handler = async (event) => {
  // Execute database flow
  const result = await dbOps();
  // Return result
  return {
    statusCode: 200,
    body: JSON.stringify("The selected sum is: " + result[0].sum)
  }
};
```

Menghubungkan ke database Amazon RDS dalam fungsi Lambda menggunakan. TypeScript

```
import { Signer } from "@aws-sdk/rds-signer";
import mysql from 'mysql2/promise';

// RDS settings
// Using '!' (non-null assertion operator) to tell the TypeScript compiler that
 the DB settings are not null or undefined,
const proxy_host_name = process.env.PROXY_HOST_NAME!
const port = parseInt(process.env.PORT!)
const db_name = process.env.DB_NAME!
const db_user_name = process.env.DB_USER_NAME!
const aws_region = process.env.AWS_REGION!


async function createAuthToken(): Promise<string> {

    // Create RDS Signer object
    const signer = new Signer({
        hostname: proxy_host_name,
        port: port,
        region: aws_region,
        username: db_user_name
    });

    // Request authorization token from RDS, specifying the username
    const token = await signer.getAuthToken();
    return token;
}

async function dbOps(): Promise<mysql.QueryResult | undefined> {
    try {
        // Obtain auth token
        const token = await createAuthToken();
        const conn = await mysql.createConnection({
            host: proxy_host_name,
            user: db_user_name,
            password: token,
            database: db_name,
            ssl: 'Amazon RDS' // Ensure you have the CA bundle for SSL connection
        });
        const [rows, fields] = await conn.execute('SELECT ? + ? AS sum', [3, 2]);
        console.log('result:', rows);
```

```
        return rows;
    }
    catch (err) {
        console.log(err);
    }
}

export const lambdaHandler = async (event: any): Promise<{ statusCode: number;
 body: string }> => {
    // Execute database flow
    const result = await dbOps();

    // Return error is result is undefined
    if (result == undefined)
        return {
            statusCode: 500,
            body: JSON.stringify(`Error with connection to DB host`)
        }

    // Return result
    return {
        statusCode: 200,
        body: JSON.stringify(`The selected sum is: ${result[0].sum}`)
    };
};
```

PHP

SDK untuk PHP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Menghubungkan ke database Amazon RDS dalam fungsi Lambda menggunakan PHP.

```
<?php
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
```

```
# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\Handler as StdHandler;
use Bref\Logger\StderrLogger;
use Aws\Rds\AuthTokenGenerator;
use Aws\Credentials\CredentialProvider;

require __DIR__ . '/vendor/autoload.php';

class Handler implements StdHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }


    private function getAuthToken(): string {
        // Define connection authentication parameters
        $dbConnection = [
            'hostname' => getenv('DB_HOSTNAME'),
            'port' => getenv('DB_PORT'),
            'username' => getenv('DB_USERNAME'),
            'region' => getenv('AWS_REGION'),
        ];

        // Create RDS AuthTokenGenerator object
        $generator = new
 AuthTokenGenerator(CredentialProvider::defaultProvider());

        // Request authorization token from RDS, specifying the username
        return $generator->createToken(
            $dbConnection['hostname'] . ':' . $dbConnection['port'],
            $dbConnection['region'],
            $dbConnection['username']
        );
    }

    private function getQueryResults() {
        // Obtain auth token
        $token = $this->getAuthToken();
```

```
        // Define connection configuration
        $connectionConfig = [
            'host' => getenv('DB_HOSTNAME'),
            'user' => getenv('DB_USERNAME'),
            'password' => $token,
            'database' => getenv('DB_NAME'),
        ];

        // Create the connection to the DB
        $conn = new PDO(

 "mysql:host={$connectionConfig['host']};dbname={$connectionConfig['database']}",
            $connectionConfig['user'],
            $connectionConfig['password'],
            [
                PDO::MYSQL_ATTR_SSL_CA => '/path/to/rds-ca-2019-root.pem',
                PDO::MYSQL_ATTR_SSL_VERIFY_SERVER_CERT => true,
            ]
        );

        // Obtain the result of the query
        $stmt = $conn->prepare('SELECT ?+? AS sum');
        $stmt->execute([3, 2]);

        return $stmt->fetch(PDO::FETCH_ASSOC);
    }

    /**
     * @param mixed $event
     * @param Context $context
     * @return array
     */
    public function handle(mixed $event, Context $context): array
    {
        $this->logger->info("Processing query");

        // Execute database flow
        $result = $this->getQueryResults();

        return [
            'sum' => $result['sum']
        ];
    }
}
```

```
$logger = new StderrLogger();
return new Handler($logger);
```

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Menghubungkan ke database Amazon RDS dalam fungsi Lambda menggunakan Python.

```
import json
import os
import boto3
import pymysql

# RDS settings
proxy_host_name = os.environ['PROXY_HOST_NAME']
port = int(os.environ['PORT'])
db_name = os.environ['DB_NAME']
db_user_name = os.environ['DB_USER_NAME']
aws_region = os.environ['AWS_REGION']


# Fetch RDS Auth Token
def get_auth_token():
    client = boto3.client('rds')
    token = client.generate_db_auth_token(
        DBHostname=proxy_host_name,
        Port=port
        DBUsername=db_user_name
        Region=aws_region
    )
    return token

def lambda_handler(event, context):
    token = get_auth_token()
```

```
    try:
        connection = pymysql.connect(
            host=proxy_host_name,
            user=db_user_name,
            password=token,
            db=db_name,
            port=port,
            ssl={'ca': 'Amazon RDS'}  # Ensure you have the CA bundle for SSL
connection
        )

        with connection.cursor() as cursor:
            cursor.execute('SELECT %s + %s AS sum', (3, 2))
            result = cursor.fetchone()

        return result

    except Exception as e:
        return (f"Error: {str(e)}")  # Return an error message if an exception
occurs
```

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Menghubungkan ke database Amazon RDS dalam fungsi Lambda menggunakan Ruby.

```
# Ruby code here.

require 'aws-sdk-rds'
require 'json'
require 'mysql2'

def lambda_handler(event:, context:)
  endpoint = ENV['DBEndpoint'] # Add the endpoint without https"
```

```
port = ENV['Port']           # 3306
user = ENV['DBUser']
region = ENV['DBRegion']     # 'us-east-1'
db_name = ENV['DBName']

credentials = Aws::Credentials.new(
  ENV['AWS_ACCESS_KEY_ID'],
  ENV['AWS_SECRET_ACCESS_KEY'],
  ENV['AWS_SESSION_TOKEN']
)
rds_client = Aws::RDS::AuthTokenGenerator.new(
  region: region,
  credentials: credentials
)

token = rds_client.auth_token(
  endpoint: endpoint+ ':' + port,
  user_name: user,
  region: region
)

begin
  conn = Mysql2::Client.new(
    host: endpoint,
    username: user,
    password: token,
    port: port,
    database: db_name,
    sslca: '/var/task/global-bundle.pem',
    sslverify: true,
    enable_cleartext_plugin: true
  )
  a = 3
  b = 2
  result = conn.query("SELECT #{a} + #{b} AS sum").first['sum']
  puts result
  conn.close
  {
    statusCode: 200,
    body: result.to_json
  }
rescue => e
  puts "Database connection failed due to #{e}"
end
```

```
end
```

Rust

SDK for Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Menghubungkan ke database Amazon RDS dalam fungsi Lambda menggunakan Rust.

```
use aws_config::BehaviorVersion;
use aws_credential_types::provider::ProvideCredentials;
use aws_sigv4::{
    http_request::{sign, SignableBody, SignableRequest, SigningSettings},
    sign::v4,
};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};
use serde_json::{json, Value};
use sqlx::postgres::PgConnectOptions;
use std::env;
use std::time::{Duration, SystemTime};

const RDS_CERTS: &[u8] = include_bytes!("global-bundle.pem");

async fn generate_rds_iam_token(
    db_hostname: &str,
    port: u16,
    db_username: &str,
) -> Result<String, Error> {
    let config = aws_config::load_defaults(BehaviorVersion::v2024_03_28()).await;

    let credentials = config
        .credentials_provider()
        .expect("no credentials provider found")
        .provide_credentials()
        .await
        .expect("unable to load credentials");
    let identity = credentials.into();
```

```
    let region = config.region().unwrap().to_string();

    let mut signing_settings = SigningSettings::default();
    signing_settings.expires_in = Some(Duration::from_secs(900));
    signing_settings.signature_location =
 aws_sigv4::http_request::SignatureLocation::QueryParams;

    let signing_params = v4::SigningParams::builder()
        .identity(&identity)
        .region(&region)
        .name("rds-db")
        .time(SystemTime::now())
        .settings(signing_settings)
        .build()?;

    let url = format!(
        "https://{db_hostname}:{port}/?Action=connect&DBUser={db_user}",
        db_hostname = db_hostname,
        port = port,
        db_user = db_username
    );

    let signable_request =
        SignableRequest::new("GET", &url, std::iter::empty(),
 SignableBody::Bytes(&[]))
            .expect("signable request");

    let (signing_instructions, _signature) =
        sign(signable_request, &signing_params.into())?.into_parts();

    let mut url = url::Url::parse(&url).unwrap();
    for (name, value) in signing_instructions.params() {
        url.query_pairs_mut().append_pair(name, &value);
    }

    let response = url.to_string().split_off("https://".len());

    Ok(response)
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    run(service_fn(handler)).await
}
```

```
async fn handler(_event: LambdaEvent<Value>) -> Result<Value, Error> {
    let db_host = env::var("DB_HOSTNAME").expect("DB_HOSTNAME must be set");
    let db_port = env::var("DB_PORT")
        .expect("DB_PORT must be set")
        .parse::<u16>()
        .expect("PORT must be a valid number");
    let db_name = env::var("DB_NAME").expect("DB_NAME must be set");
    let db_user_name = env::var("DB_USERNAME").expect("DB_USERNAME must be set");

    let token = generate_rds_iam_token(&db_host, db_port, &db_user_name).await?;

    let opts = PgConnectOptions::new()
        .host(&db_host)
        .port(db_port)
        .username(&db_user_name)
        .password(&token)
        .database(&db_name)
        .ssl_root_cert_from_pem(RDS_CERTS.to_vec())
        .ssl_mode(sqlx::postgres::PgSslMode::Require);

    let pool = sqlx::postgres::PgPoolOptions::new()
        .connect_with(opts)
        .await?;

    let result: i32 = sqlx::query_scalar("SELECT $1 + $2")
        .bind(3)
        .bind(2)
        .fetch_one(&pool)
        .await?;

    println!("Result: {:?}", result);

    Ok(json!({
        "statusCode": 200,
        "content-type": "text/plain",
        "body": format!("The selected sum is: {result}")
    }))
}
```

Memproses pemberitahuan acara dari Amazon RDS

Anda dapat menggunakan Lambda untuk memproses pemberitahuan peristiwa dari database Amazon RDS. Amazon RDS mengirimkan notifikasi ke topik Amazon Simple Notification Service (Amazon SNS), yang dapat Anda konfigurasikan untuk menjalankan Fungsi Peristiwa Lambda. Amazon SNS merangkum pesan dari Amazon RDS dalam dokumen kejadiannya sendiri dan mengirimkannya ke fungsi Anda.

Untuk informasi selengkapnya tentang mengonfigurasi database Amazon RDS untuk mengirim notifikasi, lihat Menggunakan pemberitahuan peristiwa Amazon RDS.

Example Pesan Amazon RDS di kejadian Amazon SNS

```
{
        "Records": [
          {
            "EventVersion": "1.0",
            "EventSubscriptionArn": "arn:aws:sns:us-east-2:123456789012:rds-
lambda:21be56ed-a058-49f5-8c98-aedd2564c486",
            "EventSource": "aws:sns",
            "Sns": {
              "SignatureVersion": "1",
              "Timestamp": "2023-01-02T12:45:07.000Z",
              "Signature": "tcc6faL2yUC6dgZdmrwh1Y4cGa/ebXEkAi6RibDsvpi
+tE/1+82j...65r==",
              "SigningCertUrl": "https://sns.us-east-2.amazonaws.com/
SimpleNotificationService-ac565b8b1a6c5d002d285f9598aa1d9b.pem",
              "MessageId": "95df01b4-ee98-5cb9-9903-4c221d41eb5e",
              "Message": "{\"Event Source\":\"db-instance\",\"Event Time\":\"2023-01-02
 12:45:06.000\",\"Identifier Link\":\"https://console.aws.amazon.com/rds/home?
region=eu-west-1#dbinstance:id=dbinstanceid\",\"Source ID\":\"dbinstanceid\",\"Event ID
\":\"http://docs.amazonwebservices.com/AmazonRDS/latest/UserGuide/USER_Events.html#RDS-
EVENT-0002\",\"Event Message\":\"Finished DB Instance backup\"}",
              "MessageAttributes": {},
              "Type": "Notification",
              "UnsubscribeUrl": "https://sns.us-east-2.amazonaws.com/?
Action=Unsubscribe&amp;SubscriptionArn=arn:aws:sns:us-east-2:123456789012:test-
lambda:21be56ed-a058-49f5-8c98-aedd2564c486",
              "TopicArn":"arn:aws:sns:us-east-2:123456789012:sns-lambda",
              "Subject": "RDS Notification Message"
            }
          }
        ]
```

```
    }
```

Lengkap Lambda dan Amazon RDS tutorial

- Menggunakan fungsi Lambda untuk mengakses database Amazon RDS — Dari Panduan Pengguna Amazon RDS, pelajari cara menggunakan Fungsi Peristiwa Lambda untuk menulis data ke database Amazon RDS melalui Proxy Amazon RDS. Fungsi Peristiwa Lambda Anda membaca catatan dari antrean Amazon SQS dan menulis item baru ke tabel di database Anda setiap kali pesan ditambahkan.

Pilih layanan database untuk Lambda-based aplikasi Anda

Banyak aplikasi tanpa server perlu menyimpan dan mengambil data. AWS menawarkan beberapa opsi database yang bekerja dengan Lambda Event Functions. Dua pilihan yang paling populer adalah Amazon DynamoDB, layanan database NoSQL, dan Amazon RDS, solusi database relasional tradisional. Bagian berikut menjelaskan perbedaan utama antara layanan ini saat menggunakannya dengan Lambda dan membantu Anda memilih layanan database yang tepat untuk aplikasi tanpa server Anda.

Untuk mempelajari lebih lanjut tentang layanan database lain yang ditawarkan oleh AWS, dan untuk memahami kasus penggunaan dan pengorbanannya secara lebih umum, lihat Memilih AWS layanan database. Semua layanan AWS database kompatibel dengan Lambda, tetapi tidak semuanya mungkin cocok untuk kasus penggunaan khusus Anda.

Apa pilihan Anda saat memilih layanan database dengan Lambda?

AWS menawarkan beberapa layanan database. Untuk aplikasi tanpa server, dua pilihan paling populer adalah DynamoDB dan Amazon RDS.

- DynamoDB adalah layanan database NoSQL yang dikelola sepenuhnya yang dioptimalkan untuk aplikasi tanpa server. Ini memberikan penskalaan yang mulus dan kinerja milidetik satu digit yang konsisten pada skala apa pun.
- Amazon RDS adalah layanan database relasional terkelola yang mendukung beberapa mesin database termasuk MySQL dan PostgreSQL. Ini memberikan kemampuan SQL yang akrab dengan infrastruktur terkelola.

Rekomendasi jika Anda sudah mengetahui kebutuhan Anda

Jika Anda sudah jelas tentang kebutuhan Anda, berikut adalah rekomendasi dasar kami:

Kami merekomendasikan DynamoDB untuk aplikasi tanpa server yang membutuhkan kinerja latensi rendah yang konsisten, penskalaan otomatis, dan tidak memerlukan gabungan atau transaksi yang rumit. Ini sangat cocok untuk Lambda-based aplikasi karena sifatnya yang tanpa server.

Amazon RDS adalah pilihan yang lebih baik ketika Anda membutuhkan kueri SQL yang kompleks, bergabung, atau memiliki aplikasi yang ada menggunakan database relasional. Namun, ketahuilah bahwa menghubungkan Fungsi Peristiwa Lambda ke Amazon RDS memerlukan konfigurasi tambahan dan dapat memengaruhi waktu mulai yang dingin.

Apa yang harus dipertimbangkan ketika memilih layanan database

Saat memilih antara DynamoDB dan Amazon RDS untuk aplikasi Lambda Anda, pertimbangkan faktor-faktor berikut:

- Manajemen koneksi dan start dingin
- Pola akses data
- Kompleksitas kueri
- Persyaratan konsistensi data
- Karakteristik penskalaan
- Model biaya

Dengan memahami faktor-faktor ini, Anda dapat memilih opsi yang paling memenuhi kebutuhan kasus penggunaan khusus Anda.

Manajemen koneksi dan start dingin

- DynamoDB menggunakan API HTTP untuk semua operasi. Fungsi Acara Lambda dapat membuat permintaan langsung tanpa mempertahankan koneksi, menghasilkan kinerja start dingin yang lebih baik. Setiap permintaan diautentikasi menggunakan AWS kredensil tanpa overhead koneksi.
- Amazon RDS memerlukan pengelolaan kumpulan koneksi karena menggunakan koneksi database tradisional. Ini dapat memengaruhi awal dingin karena instance Lambda baru perlu membuat koneksi. Anda harus menerapkan strategi penyatuan koneksi dan berpotensi menggunakan Amazon RDS Proxy untuk mengelola koneksi secara efektif. Perhatikan bahwa menggunakan Amazon RDS Proxy menimbulkan biaya tambahan.

Pola akses data

- DynamoDB bekerja paling baik dengan pola akses yang dikenal dan desain meja tunggal. Ini ideal untuk aplikasi Lambda yang membutuhkan akses latensi rendah yang konsisten ke data berdasarkan kunci primer atau indeks sekunder.
- Amazon RDS memberikan fleksibilitas untuk kueri kompleks dan mengubah pola akses. Ini lebih cocok ketika Fungsi Acara Lambda Anda perlu melakukan kueri yang unik dan disesuaikan atau gabungan kompleks di beberapa tabel.

Kompleksitas kueri

- DynamoDB unggul dalam operasi sederhana berbasis kunci dan pola akses yang telah ditentukan sebelumnya. Kueri kompleks harus dirancang di sekitar struktur indeks, dan gabungan harus ditangani dalam kode aplikasi.
- Amazon RDS mendukung kueri SQL yang kompleks dengan gabungan, subkueri, dan agregasi. Ini dapat menyederhanakan kode Fungsi Peristiwa Lambda Anda ketika operasi data yang kompleks diperlukan.

Persyaratan konsistensi data

- DynamoDB menawarkan opsi konsistensi akhir dan kuat, dengan konsistensi yang kuat tersedia untuk pembacaan item tunggal. Transaksi didukung tetapi dengan beberapa batasan.
- Amazon RDS memberikan kepatuhan atomisitas, konsistensi, isolasi, dan daya tahan (ACID) penuh dan dukungan transaksi yang kompleks. Jika Fungsi Peristiwa Lambda Anda memerlukan transaksi yang kompleks atau konsistensi yang kuat di beberapa catatan, Amazon RDS mungkin lebih cocok.

Karakteristik penskalaan

- DynamoDB menskalakan secara otomatis dengan beban kerja Anda. Ini dapat menangani lonjakan lalu lintas tiba-tiba dari Fungsi Peristiwa Lambda tanpa pra-penyediaan. Anda dapat menggunakan mode kapasitas sesuai permintaan untuk membayar hanya untuk apa yang Anda gunakan, sangat cocok dengan model penskalaan Lambda.
- Amazon RDS memiliki kapasitas tetap berdasarkan ukuran instans yang Anda pilih. Jika beberapa Fungsi Peristiwa Lambda mencoba terhubung secara bersamaan, Anda mungkin melebihi

kuota koneksi Anda. Anda perlu mengelola kumpulan koneksi dengan hati-hati dan berpotensi menerapkan logika coba lagi.

Model biaya

- Harga DynamoDB selaras dengan aplikasi tanpa server. Dengan kapasitas sesuai permintaan, Anda hanya membayar untuk pembacaan dan penulisan aktual yang dilakukan oleh Fungsi Acara Lambda Anda. Tidak ada biaya untuk waktu idle.
- Amazon RDS mengenakan biaya untuk instans yang sedang berjalan terlepas dari penggunaannya. Ini bisa menjadi kurang hemat biaya untuk beban kerja sporadis yang dapat menjadi tipikal dalam aplikasi tanpa server. Namun, mungkin lebih ekonomis untuk beban kerja throughput tinggi dengan penggunaan yang konsisten.

Memulai dengan layanan database pilihan Anda

Sekarang setelah Anda membaca tentang kriteria untuk memilih antara DynamoDB dan Amazon RDS dan perbedaan utama di antara keduanya, Anda dapat memilih opsi yang paling sesuai dengan kebutuhan Anda dan menggunakan sumber daya berikut untuk mulai menggunakannya.

DynamoDB

Memulai DynamoDB dengan sumber daya berikut

- Untuk pengenalan layanan DynamoDB, baca Apa itu DynamoDB? di Panduan Pengembang Amazon DynamoDB.
- Ikuti tutorial Menggunakan Lambda dengan API Gateway untuk melihat contoh penggunaan Fungsi Peristiwa Lambda untuk melakukan operasi CRUD pada tabel DynamoDB sebagai respons terhadap permintaan API.
- Baca Pemrograman dengan DynamoDB dan SDK AWS di Panduan Pengembang Amazon DynamoDB untuk mempelajari lebih lanjut tentang cara mengakses DynamoDB dari dalam Fungsi Peristiwa Lambda Anda dengan menggunakan salah satu SDK. AWS

Amazon RDS

Memulai Amazon RDS dengan sumber daya berikut

- Untuk pengenalan layanan Amazon RDS, baca Apa itu Amazon Relational Database Service (Amazon RDS)? dalam Panduan Pengguna Layanan Amazon Relational Database Service.

- Ikuti tutorial Menggunakan fungsi Lambda untuk mengakses database Amazon RDS di Panduan Pengguna Layanan Amazon Relational Database Service.
- Pelajari lebih lanjut tentang menggunakan Lambda dengan Amazon RDS dengan membaca. the section called “RDS”

Memproses pemberitahuan acara Amazon S3 dengan Lambda

Anda dapat menggunakan Lambda untuk memproses pemberitahuan kejadian dari Amazon Simple Storage Service. Amazon S3 dapat mengirim acara ke Fungsi Peristiwa Lambda saat objek dibuat atau dihapus. Anda mengonfigurasi pengaturan pemberitahuan di bucket, dan memberikan izin kepada Amazon S3 untuk memanggil fungsi di kebijakan izin berbasis sumber daya milik fungsi.

> **Warning**
>
> Jika Fungsi Peristiwa Lambda Anda menggunakan bucket yang sama yang memicunya, itu dapat menyebabkan fungsi berjalan dalam satu lingkaran. Misalnya, jika bucket memicu fungsi setiap kali suatu objek diunggah, dan fungsi mengunggah sebuah objek ke bucket, fungsi tersebut secara tidak langsung akan memicu dirinya sendiri. Untuk menghindari hal ini, gunakan dua bucket, atau konfigurasikan pemicu agar hanya diterapkan ke awalan yang digunakan untuk objek masuk.

Amazon S3 memanggil fungsi Anda secara asinkron dengan kejadian yang berisi detail tentang objek. Contoh berikut menunjukkan kejadian yang dikirimkan Amazon S3 saat paket deployment diunggah ke Amazon S3.

Example Kejadian pemberitahuan Amazon S3

```
{
  "Records": [
    {
      "eventVersion": "2.1",
      "eventSource": "aws:s3",
      "awsRegion": "us-east-2",
      "eventTime": "2019-09-03T19:37:27.192Z",
      "eventName": "ObjectCreated:Put",
      "userIdentity": {
        "principalId": "AWS:AIDAINPONIXQXHT3IKHL2"
      },
      "requestParameters": {
        "sourceIPAddress": "205.255.255.255"
      },
      "responseElements": {
        "x-amz-request-id": "D82B88E5F771F645",
        "x-amz-id-2":
 "vlR7PnpV2Ce81l0PRw6jlUpck7Jo5ZsQjryTjKlc5aLWGVHPZLj5NeC6qMa0emYBDXOo6QBU0Wo="
```

```
      },
      "s3": {
        "s3SchemaVersion": "1.0",
        "configurationId": "828aa6fc-f7b5-4305-8584-487c791949c1",
        "bucket": {
          "name": "amzn-s3-demo-bucket",
          "ownerIdentity": {
            "principalId": "A3I5XTEXAMAI3E"
          },
          "arn": "arn:aws:s3:::lambda-artifacts-deafc19498e3f2df"
        },
        "object": {
          "key": "b21b84d653bb07b05b1e6b33684dc11b",
          "size": 1305107,
          "eTag": "b21b84d653bb07b05b1e6b33684dc11b",
          "sequencer": "0C0F6F405D6ED209E1"
        }
      }
    }
  ]
}
```

Untuk memanggil fungsi Anda, Amazon S3 memerlukan izin dari kebijakan berbasis sumber daya milik fungsi. Saat Anda mengonfigurasi pemicu Amazon S3 di konsol Lambda, konsol memodifikasi kebijakan berbasis sumber daya untuk mengizinkan Amazon S3 memanggil fungsi jika nama bucket dan ID akun cocok. Jika Anda mengonfigurasi pemberitahuan di Amazon S3, Anda menggunakan API Lambda untuk memperbarui kebijakan. Anda juga dapat menggunakan API Lambda untuk memberikan izin kepada akun lain, atau membatasi izin pada alias yang ditunjuk.

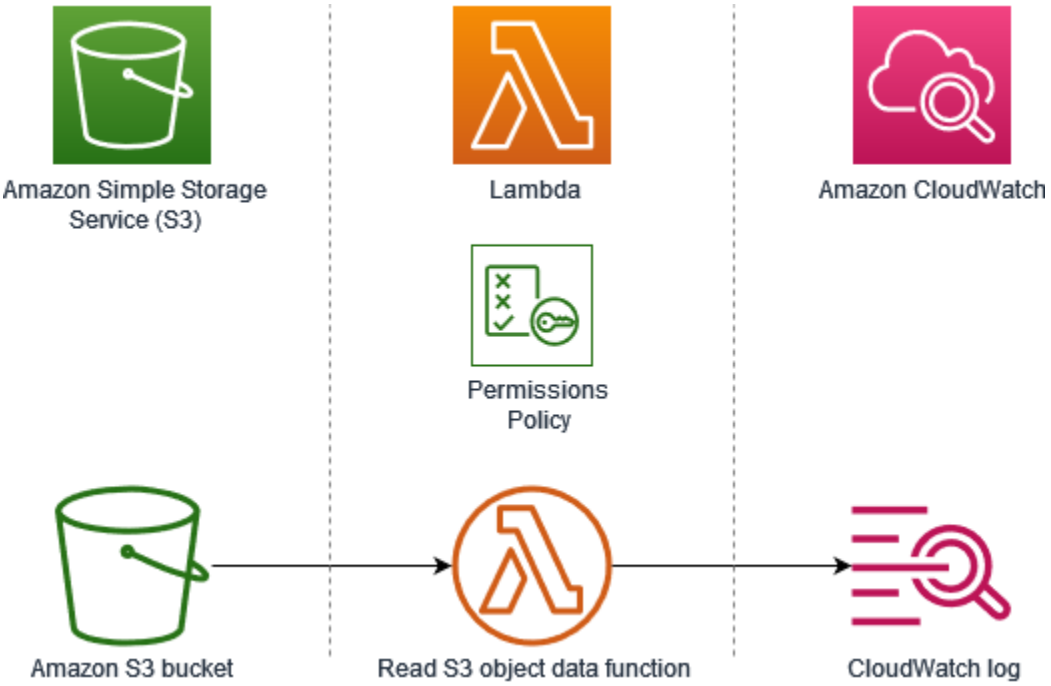
Jika fungsi Anda menggunakan AWS SDK untuk mengelola sumber daya Amazon S3, fungsi tersebut juga memerlukan izin Amazon S3 dalam peran pelaksanaannya.

Topik

- Tutorial: Menggunakan pemicu Amazon S3 untuk menjalankan Fungsi Peristiwa Lambda
- Tutorial: Menggunakan pemicu Amazon S3 untuk membuat gambar thumbnail

Tutorial: Menggunakan pemicu Amazon S3 untuk menjalankan Fungsi Peristiwa Lambda

Dalam tutorial ini, Anda menggunakan konsol untuk membuat Lambda Event Function dan mengonfigurasi pemicu untuk bucket Amazon Simple Storage Service (Amazon S3). Setiap kali Anda menambahkan objek ke bucket Amazon S3, fungsi Anda berjalan dan mengeluarkan jenis objek ke Amazon Logs. CloudWatch

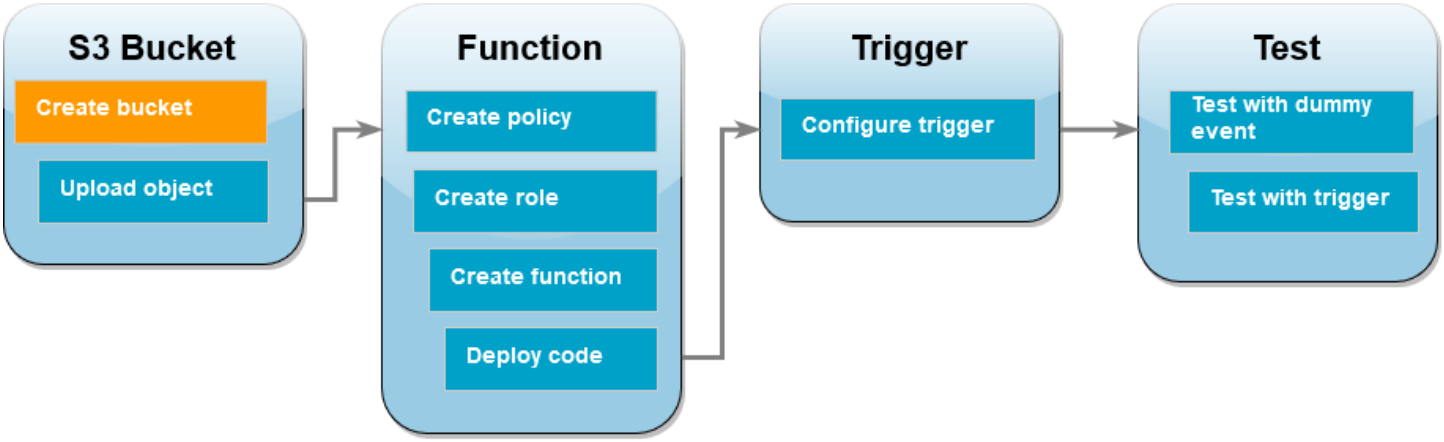


Tutorial ini menunjukkan bagaimana untuk:

1. Buat bucket Amazon S3.
2. Buat Fungsi Peristiwa Lambda yang mengembalikan jenis objek objek dalam bucket Amazon S3.
3. Konfigurasikan pemicu Lambda yang memanggil fungsi Anda saat objek diunggah ke bucket Anda.
4. Uji fungsi Anda, pertama dengan acara dummy, dan kemudian gunakan pelatuknya.

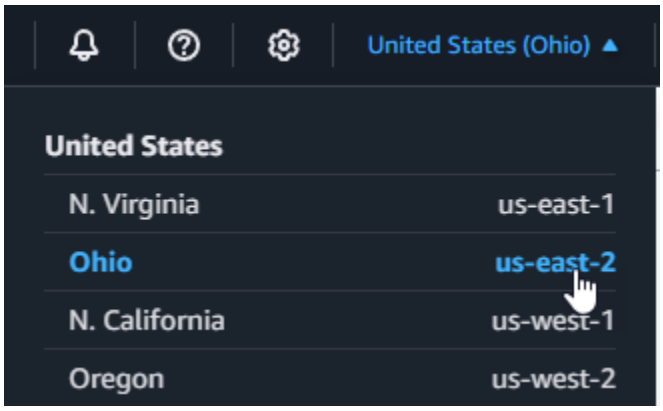
Dengan menyelesaikan langkah-langkah ini, Anda akan mempelajari cara mengonfigurasi Fungsi Peristiwa Lambda agar berjalan setiap kali objek ditambahkan atau dihapus dari bucket Amazon S3. Anda dapat menyelesaikan tutorial ini hanya dengan menggunakan Konsol Manajemen AWS.

Buat bucket Amazon S3.



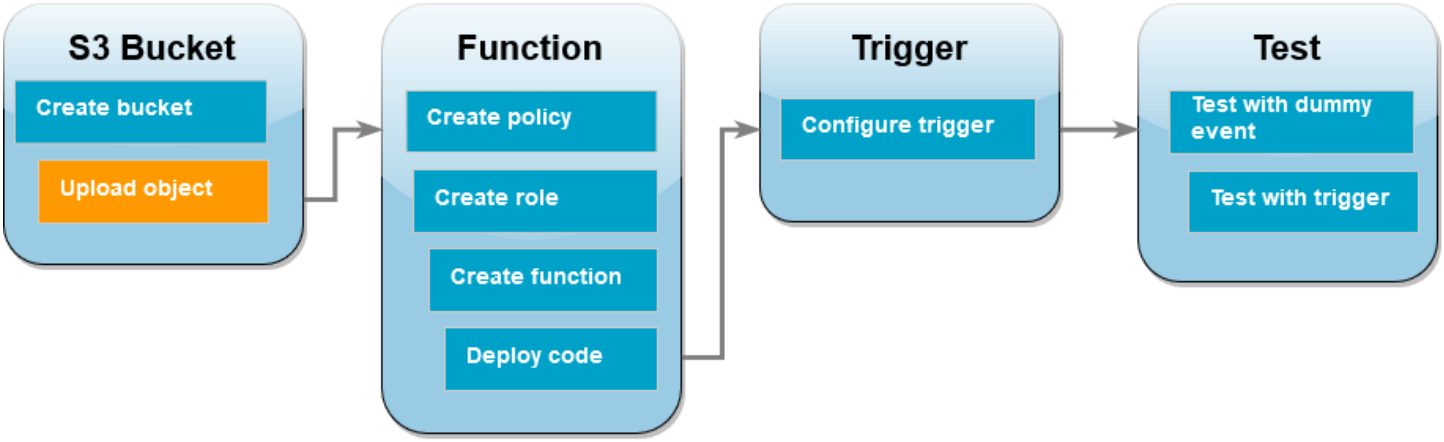
Untuk membuat bucket Amazon S3

1. Buka konsol Amazon S3 dan pilih halaman bucket tujuan umum.
2. Pilih yang Wilayah AWS paling dekat dengan lokasi geografis Anda. Anda dapat mengubah wilayah Anda menggunakan daftar drop-down di bagian atas layar. Kemudian dalam tutorial, Anda harus membuat Fungsi Acara Lambda Anda di Wilayah yang sama.



3. Pilih Buat bucket.
4. Pada Konfigurasi umum, lakukan hal berikut:
   a. Untuk jenis Bucket, pastikan Tujuan umum dipilih.
   b. Untuk nama Bucket, masukkan nama unik global yang memenuhi aturan penamaan Amazon S3 Bucket. Nama bucket hanya dapat berisi huruf kecil, angka, titik (.), dan tanda hubung (-).
5. Biarkan semua opsi lain disetel ke nilai defaultnya dan pilih Buat bucket.

Unggah objek uji ke bucket

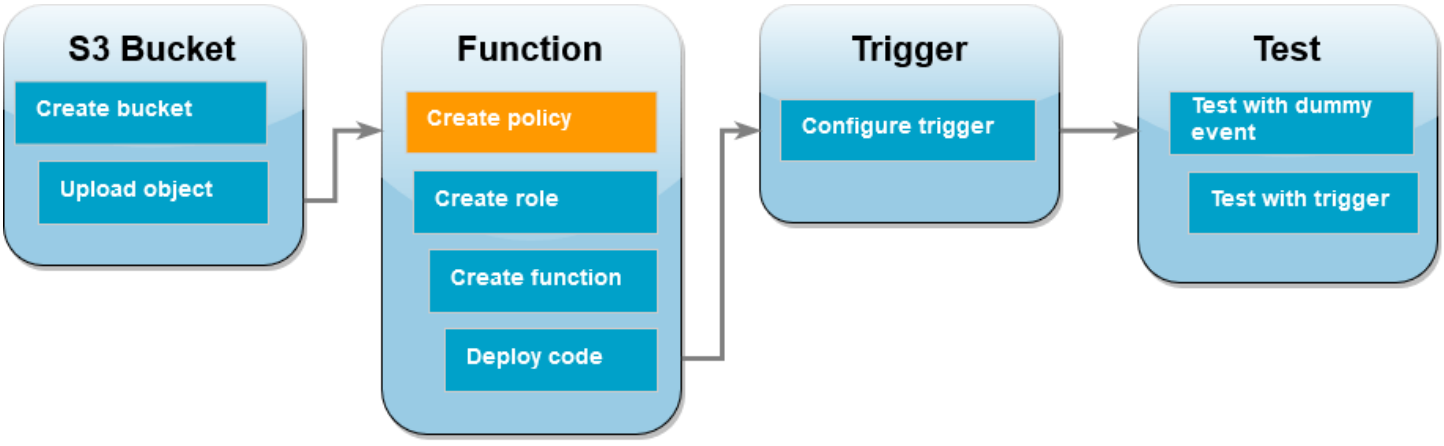


Untuk mengunggah objek uji

1. Buka halaman Bucket di konsol Amazon S3 dan pilih bucket yang Anda buat selama langkah sebelumnya.
2. Pilih Unggah.
3. Pilih Tambahkan file dan pilih objek yang ingin Anda unggah. Anda dapat memilih file apa saja (misalnya,`HappyFace.jpg`).
4. Pilih Buka, lalu pilih Unggah.

Kemudian dalam tutorial, Anda akan menguji Lambda Event Function Anda menggunakan objek ini.

Membuat kebijakan izin



Buat kebijakan izin yang memungkinkan Lambda mendapatkan objek dari bucket Amazon S3 dan menulis ke Amazon Log. CloudWatch

Untuk membuat kebijakan

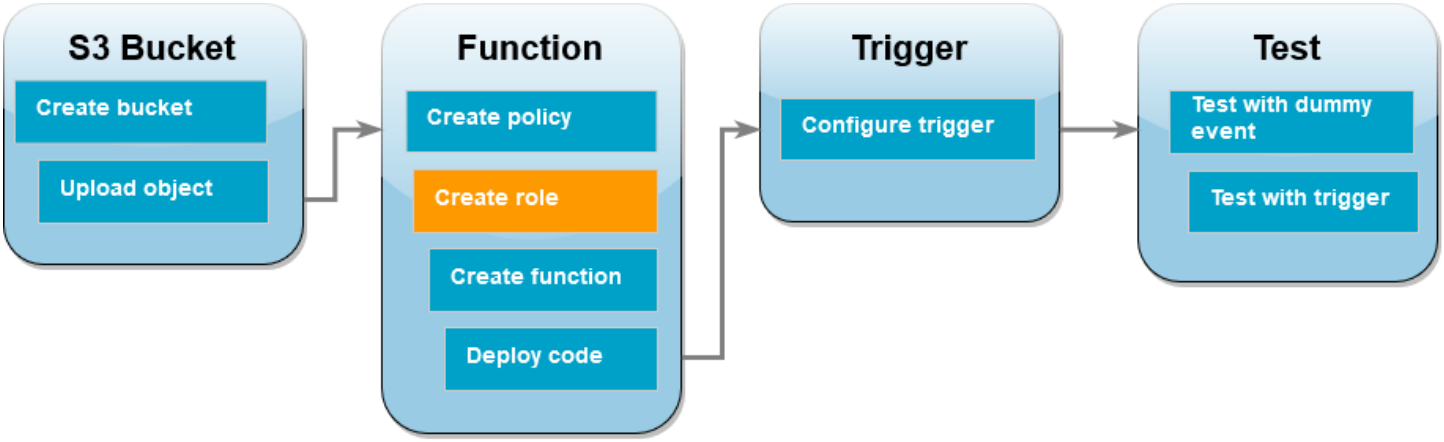
1. Buka halaman Kebijakan konsol IAM.
2. Pilih Buat Kebijakan.
3. Pilih tab JSON, lalu tempelkan kebijakan khusus berikut ke editor JSON.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents",
                "logs:CreateLogGroup",
                "logs:CreateLogStream"
            ],
            "Resource": "arn:aws:logs:*:*:*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": "arn:aws:s3:::*/*"
        }
    ]
}
```

4. Pilih Berikutnya: Tanda.
5. Pilih Berikutnya: Tinjau.
6. Di bawah Kebijakan peninjauan, untuk Nama kebijakan, masukkan**`s3-trigger-tutorial`**.
7. Pilih Buat kebijakan.

Membuat peran eksekusi

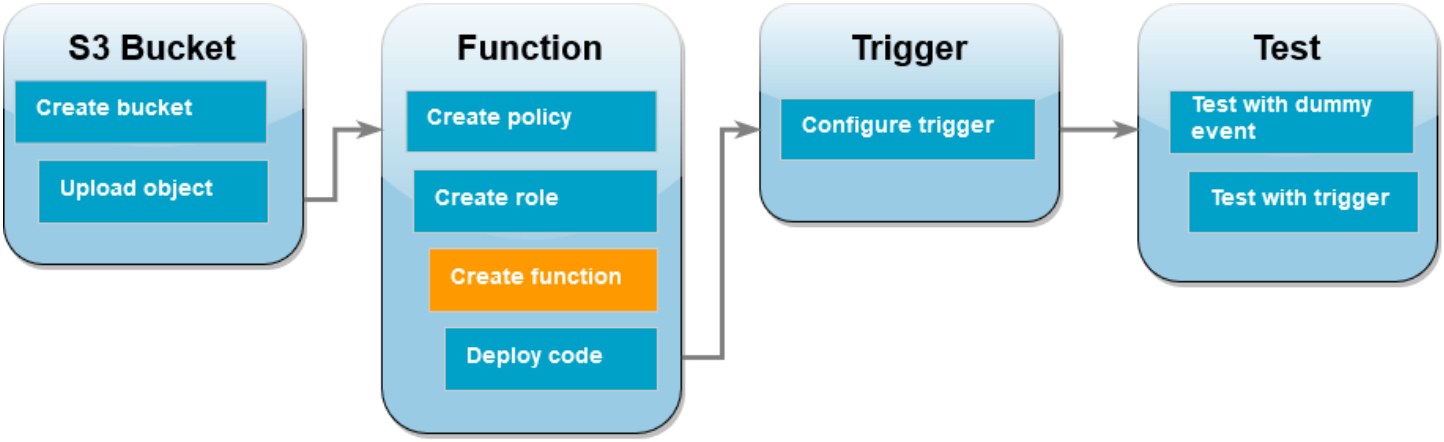


Peran eksekusi adalah peran AWS Identity and Access Management (IAM) yang memberikan izin Fungsi Peristiwa Lambda untuk mengakses dan sumber daya. Layanan AWS Pada langkah ini, buat peran eksekusi menggunakan kebijakan izin yang Anda buat di langkah sebelumnya.

Untuk membuat peran eksekusi dan melampirkan kebijakan izin khusus Anda

1. Buka halaman Peran dari konsol IAM.
2. Pilih Buat peran.
3. Untuk jenis entitas tepercaya, pilih AWS layanan, lalu untuk kasus penggunaan, pilih Lambda.
4. Pilih Berikutnya.
5. Dalam kotak pencarian kebijakan, masukkan **`s3-trigger-tutorial`**.
6. Di hasil penelusuran, pilih kebijakan yang Anda buat (`s3-trigger-tutorial`), lalu pilih Berikutnya.
7. Di bawah Rincian peran, untuk nama Peran, masukkan**`lambda-s3-trigger-role`**, lalu pilih Buat peran.

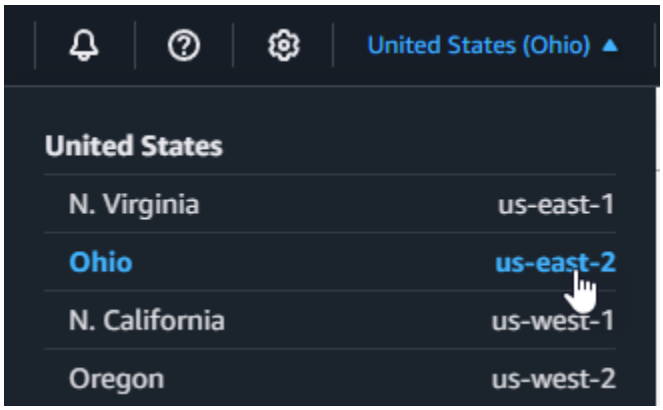
Buat Fungsi Acara Lambda



Buat Fungsi Peristiwa Lambda di konsol menggunakan runtime Python 3.14.

Untuk membuat Fungsi Acara Lambda

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pastikan Anda bekerja sama dengan saat Wilayah AWS Anda membuat bucket Amazon S3. Anda dapat mengubah Wilayah Anda menggunakan daftar drop-down di bagian atas layar.



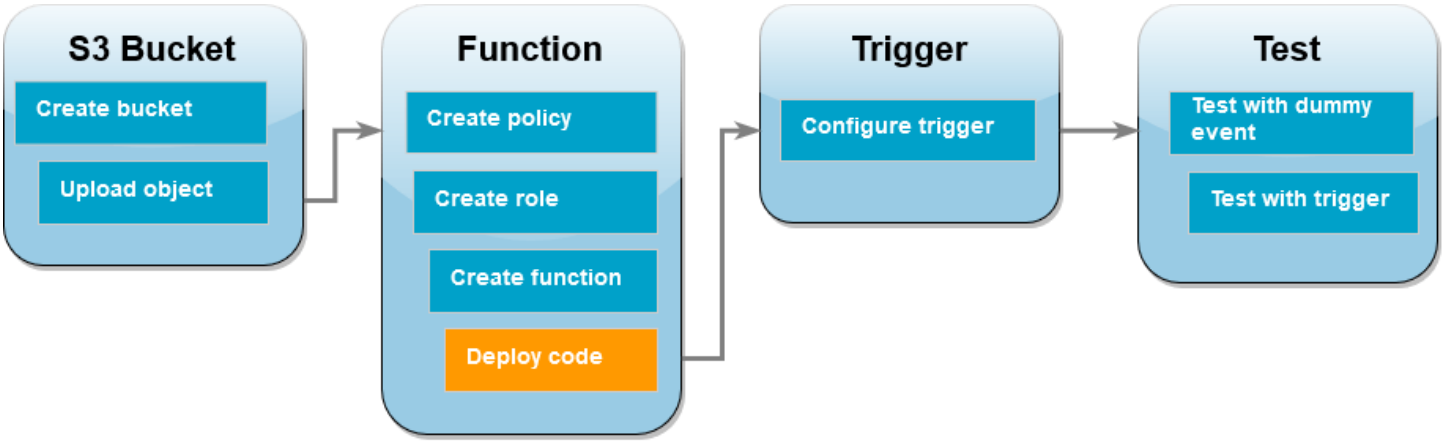
3. Pilih Buat Fungsi Acara.
4. Pilih Penulis dari awal
5. Di bagian Informasi dasar, lakukan hal berikut:
   a. Untuk nama Fungsi, masukkan `s3-trigger-tutorial`
   b. Untuk Runtime, pilih Python 3.14.
   c. Untuk Arsitektur, pilih x86_64.
6. Di tab Ubah peran eksekusi default, lakukan hal berikut:

a. Perluas tab, lalu pilih Gunakan peran yang ada.

b. Pilih yang `lambda-s3-trigger-role` Anda buat sebelumnya.

7. Pilih Buat Fungsi Acara.

Menyebarkan kode Fungsi Acara



Tutorial ini menggunakan runtime Python 3.14, tetapi kami juga menyediakan contoh file kode untuk runtime lainnya. Anda dapat memilih tab di kotak berikut untuk melihat kode runtime yang Anda minati.

Fungsi Peristiwa Lambda mengambil nama kunci objek yang diunggah dan nama bucket dari `event` parameter yang diterimanya dari Amazon S3. Fungsi kemudian menggunakan metode get_object dari AWS SDK untuk Python (Boto3) untuk mengambil metadata objek, termasuk tipe konten (tipe MIME) dari objek yang diunggah.

Untuk menyebarkan kode Event Function

1. Pilih tab Python di kotak berikut dan salin kodenya.

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Menggunakan peristiwa S3 dengan Lambda menggunakan.NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Threading.Tasks;
using Amazon.Lambda.Core;
using Amazon.S3;
using System;
using Amazon.Lambda.S3Events;
using System.Web;

// Assembly attribute to enable the Lambda function's JSON input to be
 converted into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJson

namespace S3Integration
{
    public class Function
    {
        private static AmazonS3Client _s3Client;
        public Function() : this(null)
        {
        }

        internal Function(AmazonS3Client s3Client)
        {
            _s3Client = s3Client ?? new AmazonS3Client();
        }

        public async Task<string> Handler(S3Event evt, ILambdaContext
 context)
        {
            try
            {
                if (evt.Records.Count <= 0)
                {
                    context.Logger.LogLine("Empty S3 Event received");
                    return string.Empty;
                }

                var bucket = evt.Records[0].S3.Bucket.Name;
```

```
                var key =
 HttpUtility.UrlDecode(evt.Records[0].S3.Object.Key);

                context.Logger.LogLine($"Request is for {bucket} and {key}");

                var objectResult = await _s3Client.GetObjectAsync(bucket,
 key);

                context.Logger.LogLine($"Returning {objectResult.Key}");

                return objectResult.Key;
            }
            catch (Exception e)
            {
                context.Logger.LogLine($"Error processing request -
 {e.Message}");

                return string.Empty;
            }
        }
    }
}
```

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Menggunakan peristiwa S3 dengan Lambda menggunakan Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
```

```
 "log"

 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/s3"
)

func handler(ctx context.Context, s3Event events.S3Event) error {
 sdkConfig, err := config.LoadDefaultConfig(ctx)
 if err != nil {
  log.Printf("failed to load default config: %s", err)
  return err
 }
 s3Client := s3.NewFromConfig(sdkConfig)

 for _, record := range s3Event.Records {
  bucket := record.S3.Bucket.Name
  key := record.S3.Object.URLDecodedKey
  headOutput, err := s3Client.HeadObject(ctx, &s3.HeadObjectInput{
   Bucket: &bucket,
   Key:    &key,
  })
  if err != nil {
   log.Printf("error getting head of object %s/%s: %s", bucket, key, err)
   return err
  }
  log.Printf("successfully retrieved %s/%s of type %s", bucket, key,
 *headOutput.ContentType)
 }

 return nil
}

func main() {
 lambda.Start(handler)
}
```

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Menggunakan peristiwa S3 dengan Lambda menggunakan Java.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package example;

import software.amazon.awssdk.services.s3.model.HeadObjectRequest;
import software.amazon.awssdk.services.s3.model.HeadObjectResponse;
import software.amazon.awssdk.services.s3.S3Client;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.S3Event;
import
 com.amazonaws.services.lambda.runtime.events.models.s3.S3EventNotification.S3EventNo

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class Handler implements RequestHandler<S3Event, String> {
    private static final Logger logger =
 LoggerFactory.getLogger(Handler.class);
    @Override
    public String handleRequest(S3Event s3event, Context context) {
        try {
          S3EventNotificationRecord record = s3event.getRecords().get(0);
          String srcBucket = record.getS3().getBucket().getName();
          String srcKey = record.getS3().getObject().getUrlDecodedKey();

          S3Client s3Client = S3Client.builder().build();
```

```
          HeadObjectResponse headObject = getHeadObject(s3Client, srcBucket,
 srcKey);

          logger.info("Successfully retrieved " + srcBucket + "/" + srcKey +
 " of type " + headObject.contentType());

          return "Ok";
        } catch (Exception e) {
          throw new RuntimeException(e);
        }
    }

    private HeadObjectResponse getHeadObject(S3Client s3Client, String
 bucket, String key) {
        HeadObjectRequest headObjectRequest = HeadObjectRequest.builder()
                .bucket(bucket)
                .key(key)
                .build();
        return s3Client.headObject(headObjectRequest);
    }
}
```

JavaScript

SDK untuk JavaScript (v3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara S3 dengan menggunakan JavaScript Lambda.

```
import { S3Client, HeadObjectCommand } from "@aws-sdk/client-s3";

const client = new S3Client();

export const handler = async (event, context) => {

    // Get the object from the event and show its content type
```

```
    const bucket = event.Records[0].s3.bucket.name;
    const key = decodeURIComponent(event.Records[0].s3.object.key.replace(/\
+/g, ' '));

    try {
        const { ContentType } = await client.send(new HeadObjectCommand({
            Bucket: bucket,
            Key: key,
        }));

        console.log('CONTENT TYPE:', ContentType);
        return ContentType;

    } catch (err) {
        console.log(err);
        const message = `Error getting object ${key} from bucket ${bucket}.
 Make sure they exist and your bucket is in the same region as this
 function.`;
        console.log(message);
        throw new Error(message);
    }
};
```

Mengkonsumsi acara S3 dengan menggunakan TypeScript Lambda.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { S3Event } from 'aws-lambda';
import { S3Client, HeadObjectCommand } from '@aws-sdk/client-s3';

const s3 = new S3Client({ region: process.env.AWS_REGION });

export const handler = async (event: S3Event): Promise<string | undefined> =>
 {
  // Get the object from the event and show its content type
  const bucket = event.Records[0].s3.bucket.name;
  const key = decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/
g, ' '));
  const params = {
    Bucket: bucket,
    Key: key,
  };
```

```
  try {
    const { ContentType } = await s3.send(new HeadObjectCommand(params));
    console.log('CONTENT TYPE:', ContentType);
    return ContentType;
  } catch (err) {
    console.log(err);
    const message = `Error getting object ${key} from bucket ${bucket}. Make
 sure they exist and your bucket is in the same region as this function.`;
    console.log(message);
    throw new Error(message);
  }
};
```

PHP

SDK untuk PHP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara S3 dengan Lambda menggunakan PHP.

```
<?php

use Bref\Context\Context;
use Bref\Event\S3\S3Event;
use Bref\Event\S3\S3Handler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';


class Handler extends S3Handler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
```

```
    }

    public function handleS3(S3Event $event, Context $context) : void
    {
        $this->logger->info("Processing S3 records");

        // Get the object from the event and show its content type
        $records = $event->getRecords();

        foreach ($records as $record)
        {
            $bucket = $record->getBucket()->getName();
            $key = urldecode($record->getObject()->getKey());

            try {
                $fileSize = urldecode($record->getObject()->getSize());
                echo "File Size: " . $fileSize . "\n";
                // TODO: Implement your custom processing logic here
            } catch (Exception $e) {
                echo $e->getMessage() . "\n";
                echo 'Error getting object ' . $key . ' from bucket ' .
 $bucket . '. Make sure they exist and your bucket is in the same region as
 this function.' . "\n";
                throw $e;
            }
        }
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Menggunakan peristiwa S3 dengan Lambda menggunakan Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
import json
import urllib.parse
import boto3

print('Loading function')

s3 = boto3.client('s3')


def lambda_handler(event, context):
    #print("Received event: " + json.dumps(event, indent=2))

    # Get the object from the event and show its content type
    bucket = event['Records'][0]['s3']['bucket']['name']
    key = urllib.parse.unquote_plus(event['Records'][0]['s3']['object']
['key'], encoding='utf-8')
    try:
        response = s3.get_object(Bucket=bucket, Key=key)
        print("CONTENT TYPE: " + response['ContentType'])
        return response['ContentType']
    except Exception as e:
        print(e)
        print('Error getting object {} from bucket {}. Make sure they
 exist and your bucket is in the same region as this function.'.format(key,
 bucket))
        raise e
```

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara S3 dengan Lambda menggunakan Ruby.

```
require 'json'
require 'uri'
require 'aws-sdk'

puts 'Loading function'

def lambda_handler(event:, context:)
  s3 = Aws::S3::Client.new(region: 'region') # Your AWS region
  # puts "Received event: #{JSON.dump(event)}"

  # Get the object from the event and show its content type
  bucket = event['Records'][0]['s3']['bucket']['name']
  key = URI.decode_www_form_component(event['Records'][0]['s3']['object']
['key'], Encoding::UTF_8)
  begin
    response = s3.get_object(bucket: bucket, key: key)
    puts "CONTENT TYPE: #{response.content_type}"
    return response.content_type
  rescue StandardError => e
    puts e.message
    puts "Error getting object #{key} from bucket #{bucket}. Make sure they
 exist and your bucket is in the same region as this function."
    raise e
  end
end
```

Rust

SDK for Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Menggunakan peristiwa S3 dengan Lambda menggunakan Rust.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::event::s3::S3Event;
use aws_sdk_s3::{Client};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};


/// Main function
#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        .with_target(false)
        .without_time()
        .init();

    // Initialize the AWS SDK for Rust
    let config = aws_config::load_from_env().await;
    let s3_client = Client::new(&config);

    let res = run(service_fn(|request: LambdaEvent<S3Event>| {
        function_handler(&s3_client, request)
    })).await;

    res
}

async fn function_handler(
    s3_client: &Client,
```

```
    evt: LambdaEvent<S3Event>
) -> Result<(), Error> {
    tracing::info!(records = ?evt.payload.records.len(), "Received request
 from SQS");

    if evt.payload.records.len() == 0 {
        tracing::info!("Empty S3 event received");
    }

    let bucket =
 evt.payload.records[0].s3.bucket.name.as_ref().expect("Bucket name to
 exist");
    let key = evt.payload.records[0].s3.object.key.as_ref().expect("Object
 key to exist");

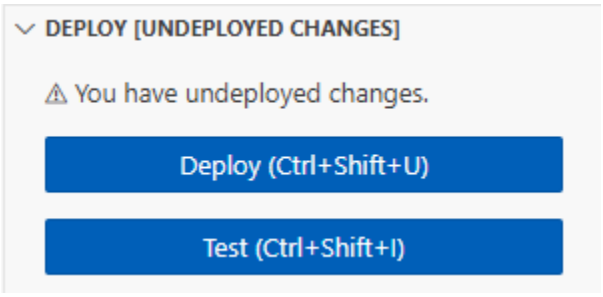
    tracing::info!("Request is for {} and object {}", bucket, key);

    let s3_get_object_result = s3_client
        .get_object()
        .bucket(bucket)
        .key(key)
        .send()
        .await;

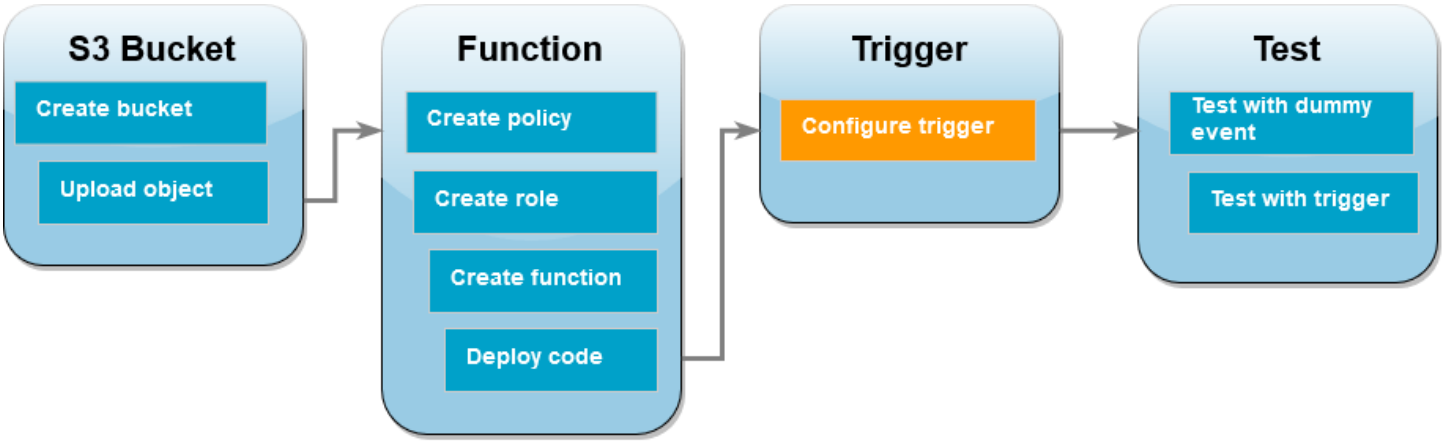
    match s3_get_object_result {
        Ok(_) => tracing::info!("S3 Get Object success, the s3GetObjectResult
 contains a 'body' property of type ByteStream"),
        Err(_) => tracing::info!("Failure with S3 Get Object request")
    }

    Ok(())
}
```

2. Di panel Sumber kode di konsol Lambda, tempelkan kode ke editor kode, ganti kode yang dibuat Lambda.
3. Di bagian DEPLOY, pilih Deploy untuk memperbarui kode fungsi Anda:

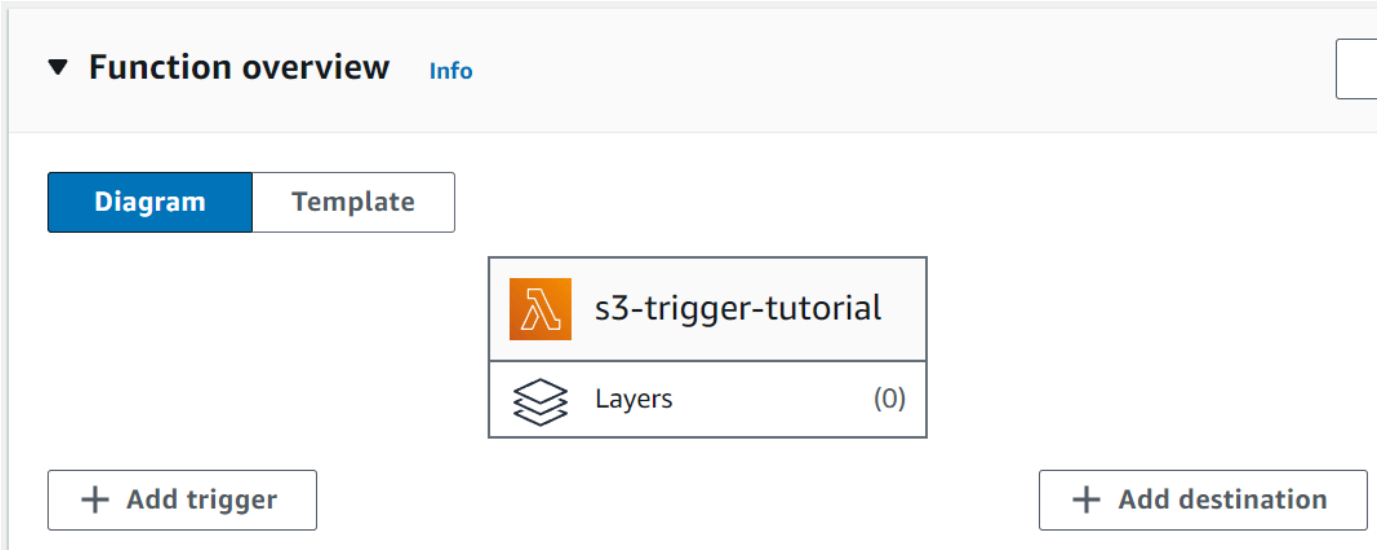


Buat pemicu Amazon S3



Untuk membuat pemicu Amazon S3

1. Di panel Ikhtisar fungsi, pilih Tambah pemicu.

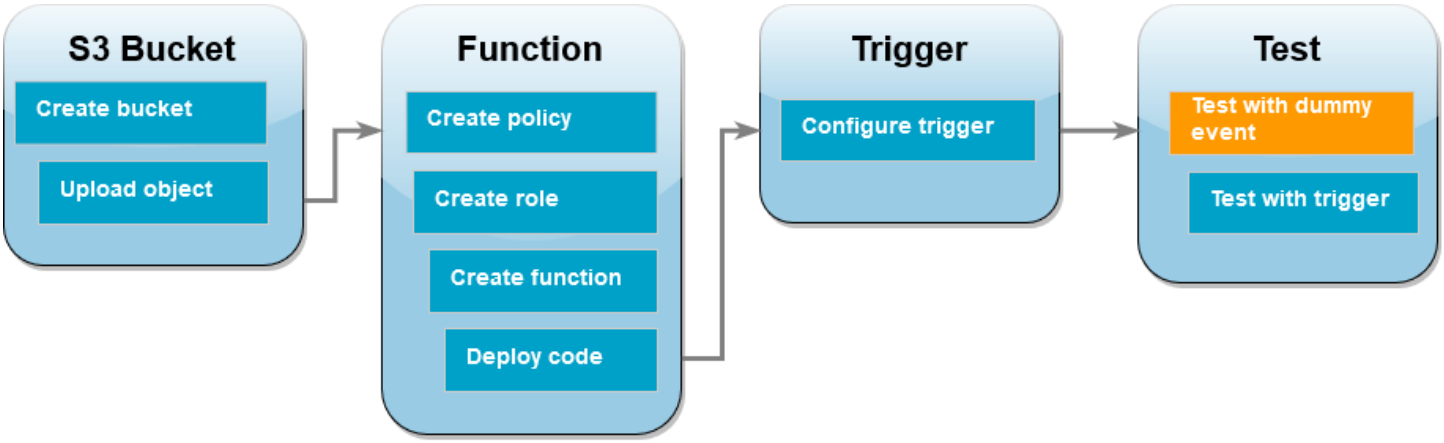


2. Pilih S3.
3. Di bawah Bucket, pilih bucket yang Anda buat sebelumnya di tutorial.

4. Di bawah Jenis acara, pastikan bahwa Semua peristiwa pembuatan objek dipilih.
5. Di bawah Pemanggilan rekursif, pilih kotak centang untuk mengetahui bahwa tidak disarankan menggunakan bucket Amazon S3 yang sama untuk input dan output.
6. Pilih Tambahkan.

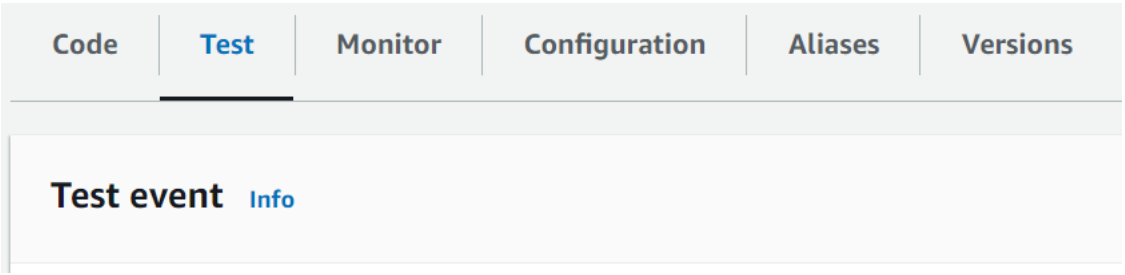
> **Note**
>
> Saat Anda membuat pemicu Amazon S3 untuk Fungsi Peristiwa Lambda menggunakan konsol Lambda, Amazon S3 mengonfigurasi pemberitahuan peristiwa di bucket yang Anda tentukan. Sebelum mengonfigurasi pemberitahuan peristiwa ini, Amazon S3 melakukan serangkaian pemeriksaan untuk mengonfirmasi bahwa tujuan acara ada dan memiliki kebijakan IAM yang diperlukan. Amazon S3 juga melakukan pengujian ini pada pemberitahuan acara lain yang dikonfigurasi untuk bucket itu.
> Karena pemeriksaan ini, jika bucket sebelumnya telah mengonfigurasi tujuan peristiwa untuk sumber daya yang tidak ada lagi, atau untuk sumber daya yang tidak memiliki kebijakan izin yang diperlukan, Amazon S3 tidak akan dapat membuat notifikasi peristiwa baru. Anda akan melihat pesan galat berikut yang menunjukkan bahwa pemicu Anda tidak dapat dibuat:
>
> ```
> An error occurred when creating the trigger: Unable to validate the following
>  destination configurations.
> ```
>
> Anda dapat melihat kesalahan ini jika sebelumnya mengonfigurasi pemicu untuk Fungsi Peristiwa Lambda lain menggunakan bucket yang sama, dan sejak itu Anda telah menghapus fungsi atau memodifikasi kebijakan izinnya.

Uji Fungsi Acara Lambda Anda dengan acara dummy



Untuk menguji Fungsi Peristiwa Lambda dengan acara dummy

1. Di halaman konsol Lambda untuk fungsi Anda, pilih tab Uji.



2. Untuk Nama peristiwa, masukkan `MyTestEvent`.
3. Di Event JSON, tempel peristiwa pengujian berikut. Pastikan untuk mengganti nilai-nilai ini:
   - Ganti `us-east-1` dengan wilayah tempat Anda membuat bucket Amazon S3.
   - Ganti kedua instance `amzn-s3-demo-bucket` dengan nama bucket Amazon S3 Anda sendiri.
   - Ganti `test%2FKey` dengan nama objek pengujian yang Anda unggah ke bucket sebelumnya (misalnya,`HappyFace.jpg`).

```
{
  "Records": [
    {
      "eventVersion": "2.0",
      "eventSource": "aws:s3",
      "awsRegion": "us-east-1",
      "eventTime": "1970-01-01T00:00:00.000Z",
      "eventName": "ObjectCreated:Put",
      "userIdentity": {
        "principalId": "EXAMPLE"
      },
      "requestParameters": {
        "sourceIPAddress": "127.0.0.1"
      },
      "responseElements": {
        "x-amz-request-id": "EXAMPLE123456789",
        "x-amz-id-2": "EXAMPLE123/5678abcdefghijklambdaisawesome/
mnopqrstuvwxyzABCDEFGH"
      },
      "s3": {
        "s3SchemaVersion": "1.0",
```

```
        "configurationId": "testConfigRule",
        "bucket": {
          "name": "amzn-s3-demo-bucket",
          "ownerIdentity": {
            "principalId": "EXAMPLE"
          },
          "arn": "arn:aws:s3:::amzn-s3-demo-bucket"
        },
        "object": {
          "key": "test%2Fkey",
          "size": 1024,
          "eTag": "0123456789abcdef0123456789abcdef",
          "sequencer": "0A1B2C3D4E5F678901"
        }
      }
    }
  ]
}
```

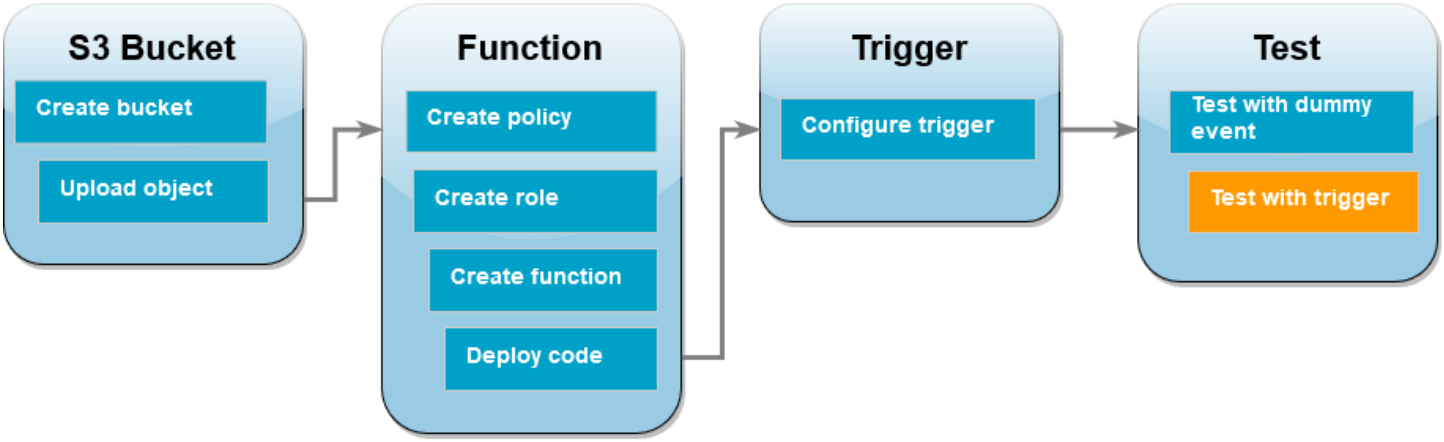
4. Pilih Simpan.
5. Pilih Uji.
6. Jika fungsi Anda berjalan dengan sukses, Anda akan melihat output yang mirip dengan berikut ini di tab Hasil eksekusi.

```
Response
"image/jpeg"

Function Logs
START RequestId: 12b3cae7-5f4e-415e-93e6-416b8f8b66e6 Version: $LATEST
2021-02-18T21:40:59.280Z    12b3cae7-5f4e-415e-93e6-416b8f8b66e6    INFO    INPUT
 BUCKET AND KEY:  { Bucket: 'amzn-s3-demo-bucket', Key: 'HappyFace.jpg' }
2021-02-18T21:41:00.215Z    12b3cae7-5f4e-415e-93e6-416b8f8b66e6    INFO    CONTENT
 TYPE: image/jpeg
END RequestId: 12b3cae7-5f4e-415e-93e6-416b8f8b66e6
REPORT RequestId: 12b3cae7-5f4e-415e-93e6-416b8f8b66e6    Duration: 976.25 ms
 Billed Duration: 977 ms    Memory Size: 128 MB    Max Memory Used: 90 MB    Init
 Duration: 430.47 ms

Request ID
12b3cae7-5f4e-415e-93e6-416b8f8b66e6
```

Uji Fungsi Peristiwa Lambda dengan pemicu Amazon S3



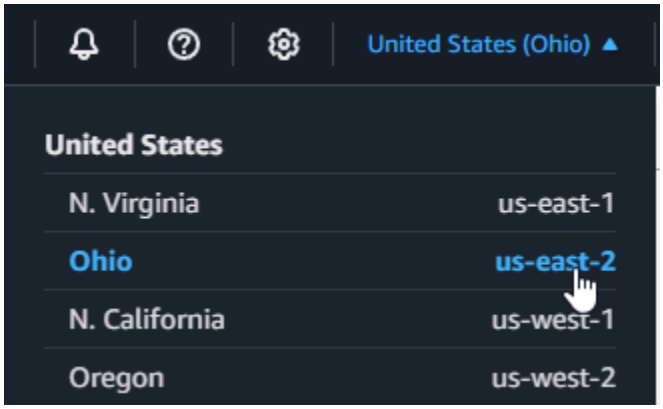
Untuk menguji fungsi Anda dengan pemicu yang dikonfigurasi, unggah objek ke bucket Amazon S3 menggunakan konsol. Untuk memverifikasi bahwa Fungsi Peristiwa Lambda Anda berjalan seperti yang diharapkan, gunakan CloudWatch Log untuk melihat output fungsi Anda.

Untuk mengunggah objek ke bucket Amazon S3 Anda

1. Buka halaman Bucket di konsol Amazon S3 dan pilih bucket yang Anda buat sebelumnya.
2. Pilih Unggah.
3. Pilih Tambahkan file dan gunakan pemilih file untuk memilih objek yang ingin Anda unggah. Objek ini dapat berupa file apa pun yang Anda pilih.
4. Pilih Buka, lalu pilih Unggah.

Untuk memverifikasi pemanggilan Fungsi Peristiwa menggunakan Log CloudWatch

1. Buka konsol CloudWatch.
2. Pastikan Anda bekerja sama dengan yang Wilayah AWS Anda buat di Lambda Event Function. Anda dapat mengubah Wilayah Anda menggunakan daftar drop-down di bagian atas layar.



3. Pilih Log, lalu pilih Grup log.
4. Pilih grup log untuk fungsi Anda (`/aws/lambda/s3-trigger-tutorial`).
5. Di bawah Aliran log, pilih aliran log terbaru.
6. Jika fungsi Anda dipanggil dengan benar sebagai respons terhadap pemicu Amazon S3, Anda akan melihat output yang mirip dengan berikut ini. Yang `CONTENT TYPE` Anda lihat tergantung pada jenis file yang Anda unggah ke bucket Anda.

```
2022-05-09T23:17:28.702Z 0cae7f5a-b0af-4c73-8563-a3430333cc10 INFO CONTENT
 TYPE: image/jpeg
```

Bersihkan sumber daya Anda

Sekarang Anda dapat menghapus sumber daya yang Anda buat untuk tutorial ini, kecuali Anda ingin mempertahankannya. Dengan menghapus AWS sumber daya yang tidak lagi Anda gunakan, Anda mencegah tagihan yang tidak perlu ke Anda Akun AWS.

Untuk menghapus Fungsi Acara Lambda

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih Fungsi Acara yang Anda buat.
3. Pilih Tindakan, Hapus.
4. Ketik **`confirm`** kolom input teks dan pilih Hapus.

Untuk menghapus peran eksekusi

1. Buka halaman Peran dari konsol IAM.
2. Pilih peran eksekusi yang Anda buat.

3. Pilih Hapus.
4. Masukkan nama peran di bidang input teks dan pilih Hapus.

Untuk menghapus bucket S3

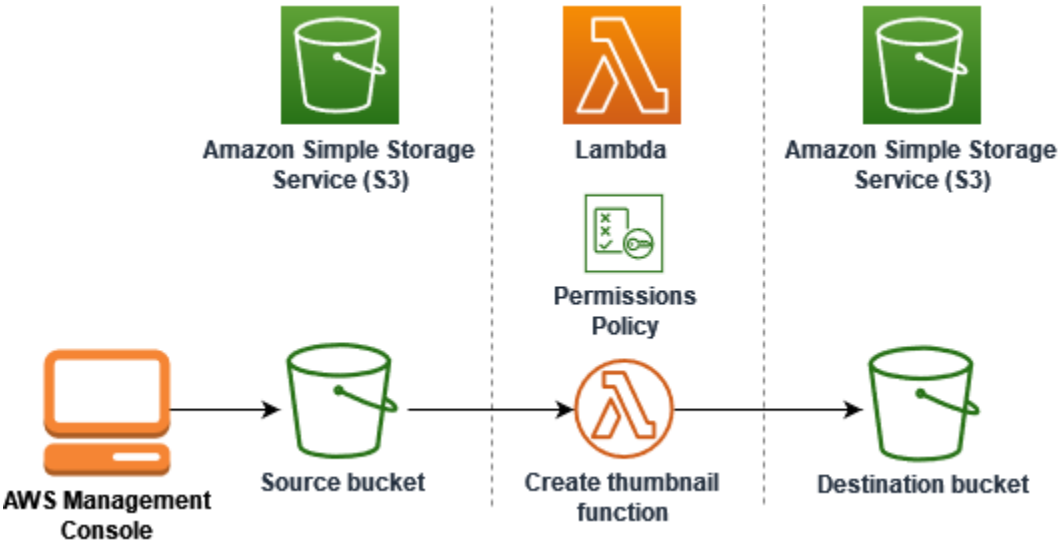
1. Buka konsol Amazon S3.
2. Pilih bucket yang Anda buat.
3. Pilih Hapus.
4. Masukkan nama ember di bidang input teks.
5. Pilih Hapus bucket.

Langkah selanjutnya

DiTutorial: Menggunakan pemicu Amazon S3 untuk membuat gambar thumbnail, pemicu Amazon S3 memanggil fungsi yang membuat gambar mini untuk setiap file gambar yang diunggah ke bucket. Tutorial ini membutuhkan pengetahuan domain Lambda AWS dan tingkat moderat. Ini menunjukkan cara membuat sumber daya menggunakan AWS Command Line Interface (AWS CLI) dan cara membuat paket penyebaran arsip file.zip untuk fungsi dan dependensinya.

Tutorial: Menggunakan pemicu Amazon S3 untuk membuat gambar thumbnail

Dalam tutorial ini, Anda membuat dan mengonfigurasi Lambda Event Function yang mengubah ukuran gambar yang ditambahkan ke bucket Amazon Simple Storage Service (Amazon S3). Saat menambahkan file gambar ke bucket, Amazon S3 akan memanggil Fungsi Acara Lambda. Fungsi tersebut kemudian membuat versi thumbnail gambar dan mengeluarkannya ke bucket Amazon S3 yang berbeda.



Untuk menyelesaikan tutorial ini, Anda melakukan langkah-langkah berikut:

1. Buat bucket Amazon S3 sumber dan tujuan dan unggah gambar sampel.
2. Buat Fungsi Peristiwa Lambda yang mengubah ukuran gambar dan mengeluarkan thumbnail ke bucket Amazon S3.
3. Konfigurasikan pemicu Lambda yang memanggil fungsi Anda saat objek diunggah ke bucket sumber Anda.
4. Uji fungsi Anda, pertama dengan acara dummy, lalu dengan mengunggah gambar ke bucket sumber Anda.

Dengan menyelesaikan langkah-langkah ini, Anda akan mempelajari cara menggunakan Lambda untuk menjalankan tugas pemrosesan file pada objek yang ditambahkan ke bucket Amazon S3. Anda dapat menyelesaikan tutorial ini menggunakan AWS Command Line Interface (AWS CLI) atau Konsol Manajemen AWS.

Jika Anda mencari contoh sederhana untuk mempelajari cara mengonfigurasi pemicu Amazon S3 untuk Lambda, Anda dapat mencoba Tutorial: Menggunakan pemicu Amazon S3 untuk menjalankan fungsi Lambda.

Topik

- Prasyarat
- Buat dua ember Amazon S3
- Unggah gambar uji ke bucket sumber Anda

- Membuat kebijakan izin
- Membuat peran eksekusi
- Buat paket penyebaran Fungsi Acara
- Buat Fungsi Acara Lambda
- Konfigurasikan Amazon S3 untuk menjalankan Fungsi Acara
- Uji Fungsi Acara Lambda Anda dengan acara dummy
- Uji Fungsi Acara Anda menggunakan pemicu Amazon S3
- Bersihkan sumber daya Anda

Prasyarat

Jika Anda ingin menggunakan AWS CLI untuk menyelesaikan tutorial, instal versi terbaru dari AWS Command Line Interface.

Untuk kode Lambda Event Function Anda, Anda dapat menggunakan Python atau. Node.js Instal alat dukungan bahasa dan manajer paket untuk bahasa yang ingin Anda gunakan.

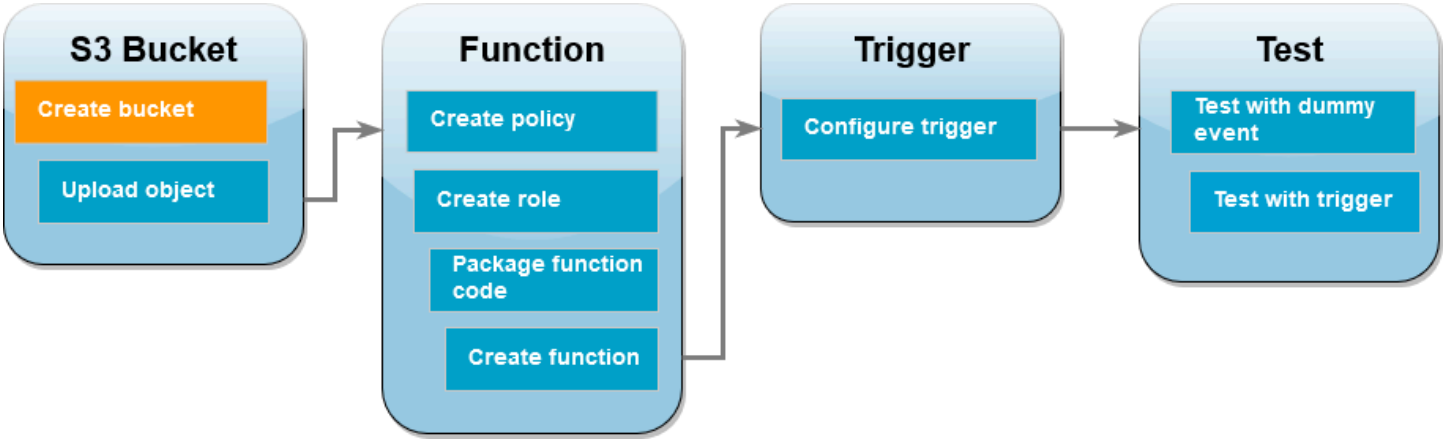
Instal AWS Command Line Interface

Jika Anda belum menginstal AWS Command Line Interface, ikuti langkah-langkah di Menginstal atau memperbarui versi terbaru AWS CLI untuk menginstalnya.

Tutorial ini membutuhkan terminal baris perintah atau shell untuk menjalankan perintah. Di Linux dan macOS, gunakan shell dan manajer paket pilihan Anda.

> **Note**
>
> Di Windows, beberapa perintah Bash CLI yang biasa Anda gunakan dengan Lambda (`zip`seperti) tidak didukung oleh terminal bawaan sistem operasi. Untuk mendapatkan Windows-integrated versi Ubuntu dan Bash, instal Subsistem Windows untuk Linux.

Buat dua ember Amazon S3

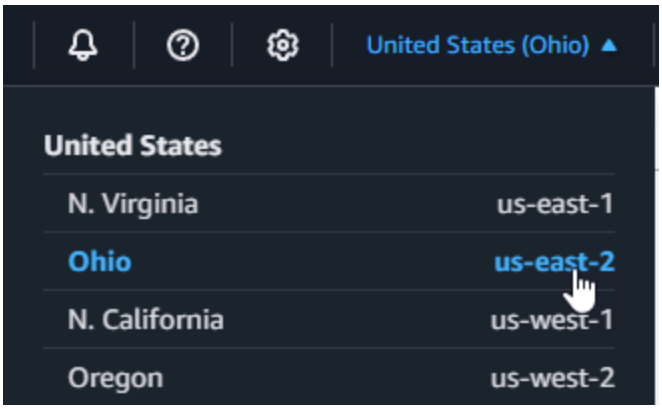


Pertama buat dua ember Amazon S3. Bucket pertama adalah bucket sumber tempat Anda mengunggah gambar Anda. Bucket kedua digunakan oleh Lambda untuk menyimpan thumbnail yang diubah ukurannya saat Anda menjalankan fungsi.

Konsol Manajemen AWS

Untuk membuat bucket Amazon S3 (konsol)

1. Buka konsol Amazon S3 dan pilih halaman bucket tujuan umum.
2. Pilih yang Wilayah AWS paling dekat dengan lokasi geografis Anda. Anda dapat mengubah wilayah Anda menggunakan daftar drop-down di bagian atas layar. Kemudian dalam tutorial, Anda harus membuat Fungsi Acara Lambda Anda di Wilayah yang sama.



3. Pilih Buat bucket.
4. Pada Konfigurasi umum, lakukan hal berikut:

   a. Untuk jenis Bucket, pastikan Tujuan umum dipilih.

b. Untuk nama Bucket, masukkan nama unik global yang memenuhi aturan penamaan Amazon S3 Bucket. Nama bucket hanya dapat berisi huruf kecil, angka, titik (.), dan tanda hubung (-).

5. Biarkan semua opsi lain disetel ke nilai defaultnya dan pilih Buat bucket.
6. Ulangi langkah 1 hingga 5 untuk membuat bucket tujuan Anda. Untuk nama Bucket**amzn-s3-demo-source-bucket-resized**, masukkan, di **amzn-s3-demo-source-bucket** mana nama bucket sumber yang baru saja Anda buat.

AWS CLI

Untuk membuat ember Amazon S3 (AWS CLI)

1. Jalankan perintah CLI berikut untuk membuat bucket sumber Anda. Nama yang Anda pilih untuk bucket Anda harus unik secara global dan ikuti aturan penamaan Amazon S3 Bucket. Nama hanya dapat berisi huruf kecil, angka, titik (.), dan tanda hubung (-). Untuk `region` dan`LocationConstraint`, pilih yang paling Wilayah AWSdekat dengan lokasi geografis Anda.

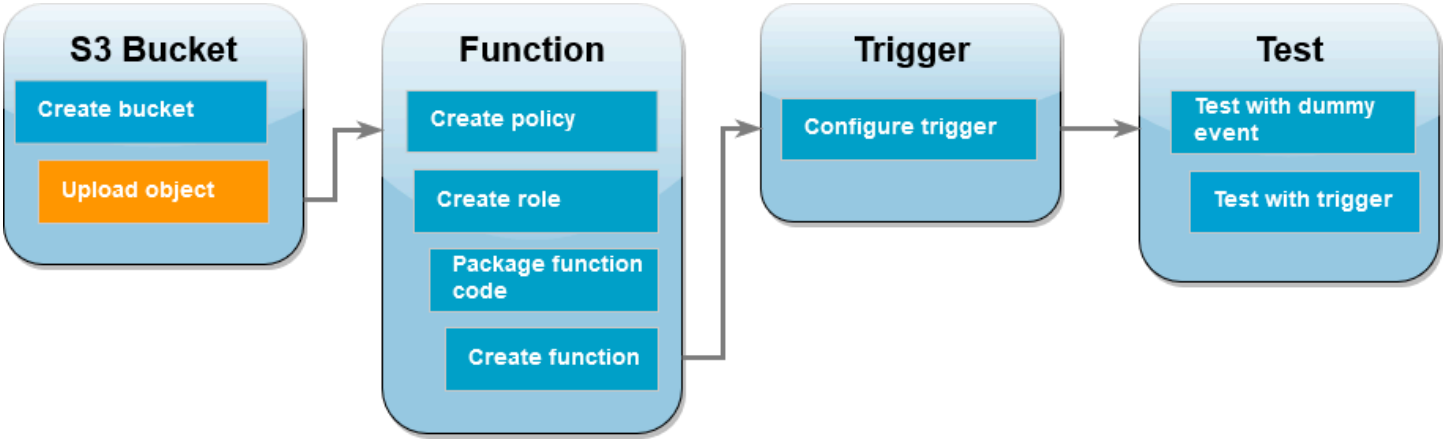
```
aws s3api create-bucket --bucket amzn-s3-demo-source-bucket --region us-east-1 \
--create-bucket-configuration LocationConstraint=us-east-1
```

Kemudian dalam tutorial, Anda harus membuat Lambda Event Function Anda Wilayah AWS sama dengan bucket sumber Anda, jadi catat wilayah yang Anda pilih.

2. Jalankan perintah berikut untuk membuat bucket tujuan Anda. Untuk nama bucket, Anda harus menggunakan**amzn-s3-demo-source-bucket-resized**, di **amzn-s3-demo-source-bucket** mana nama bucket sumber yang Anda buat di langkah 1. Untuk `region` dan`LocationConstraint`, pilih yang sama dengan yang Wilayah AWS Anda gunakan untuk membuat bucket sumber Anda.

```
aws s3api create-bucket --bucket amzn-s3-demo-source-bucket-resized --region us-
east-1 \
--create-bucket-configuration LocationConstraint=us-east-1
```

Unggah gambar uji ke bucket sumber Anda



Kemudian dalam tutorial, Anda akan menguji Lambda Event Function Anda dengan memanggilnya menggunakan atau konsol Lambda. AWS CLI Untuk mengonfirmasi bahwa fungsi Anda beroperasi dengan benar, bucket sumber Anda harus berisi gambar uji. Gambar ini dapat berupa file JPG atau PNG yang Anda pilih.

Konsol Manajemen AWS

Untuk mengunggah gambar uji ke bucket sumber Anda (konsol)

1. Buka halaman Bucket konsol Amazon S3.
2. Pilih bucket sumber yang Anda buat di langkah sebelumnya.
3. Pilih Unggah.
4. Pilih Tambahkan file dan gunakan pemilih file untuk memilih objek yang ingin Anda unggah.
5. Pilih Buka, lalu pilih Unggah.

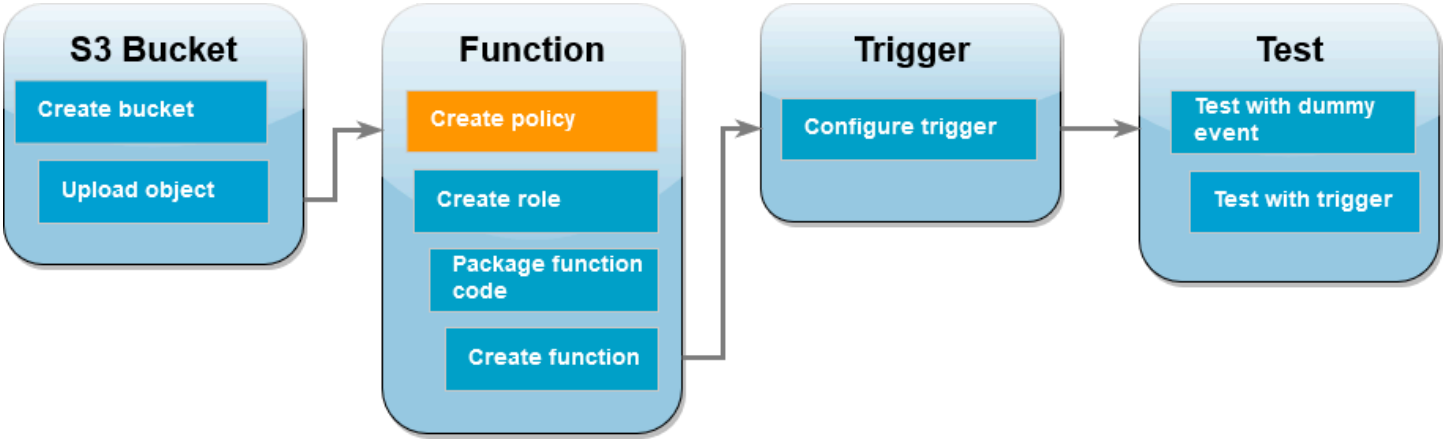
AWS CLI

Untuk mengunggah gambar uji ke bucket sumber Anda (AWS CLI)

- Dari direktori yang berisi gambar yang ingin Anda unggah, jalankan perintah CLI berikut. Ganti `--bucket` parameter dengan nama bucket sumber Anda. Untuk `--body` parameter `--key` dan, gunakan nama file gambar pengujian Anda.

```
aws s3api put-object --bucket amzn-s3-demo-source-bucket --key HappyFace.jpg --
body ./HappyFace.jpg
```

Membuat kebijakan izin



Langkah pertama dalam membuat Fungsi Peristiwa Lambda Anda adalah membuat kebijakan izin. Kebijakan ini memberi fungsi Anda izin yang diperlukan untuk mengakses AWS sumber daya lain. Untuk tutorial ini, kebijakan memberikan izin baca dan tulis Lambda untuk bucket Amazon S3 dan memungkinkannya menulis ke Amazon Log. CloudWatch

Konsol Manajemen AWS

Untuk membuat kebijakan (konsol)

1. Buka halaman Kebijakan konsol AWS Identity and Access Management (IAM).
2. Pilih Buat kebijakan.
3. Pilih tab JSON, lalu tempelkan kebijakan khusus berikut ke editor JSON.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents",
                "logs:CreateLogGroup",
                "logs:CreateLogStream"
            ],
            "Resource": "arn:aws:logs:*:*:*"
        },
        {
            "Effect": "Allow",
            "Action": [
```

```
                "s3:GetObject"
            ],
            "Resource": "arn:aws:s3:::*/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject"
            ],
            "Resource": "arn:aws:s3:::*/*"
        }
    ]
}
```

4. Pilih Berikutnya.
5. Di bawah Detail kebijakan, untuk nama Kebijakan, masukkan**`LambdaS3Policy`**.
6. Pilih Buat kebijakan.

AWS CLI

Untuk membuat kebijakan (AWS CLI)

1. Simpan JSON berikut dalam file bernama`policy.json`.

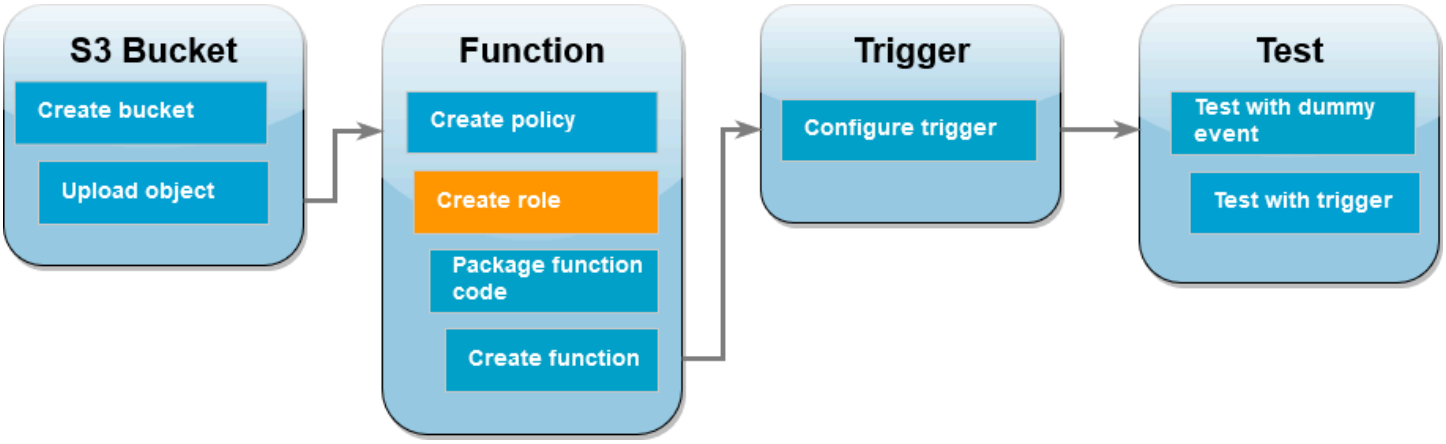
```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents",
                "logs:CreateLogGroup",
                "logs:CreateLogStream"
            ],
            "Resource": "arn:aws:logs:*:*:*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
```

```
            "Resource": "arn:aws:s3:::*/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject"
            ],
            "Resource": "arn:aws:s3:::*/*"
        }
    ]
}
```

2. Dari direktori tempat Anda menyimpan dokumen kebijakan JSON, jalankan perintah CLI berikut.

```
aws iam create-policy --policy-name LambdaS3Policy --policy-document file://
policy.json
```

Membuat peran eksekusi



Peran eksekusi adalah peran IAM yang memberikan izin Fungsi Peristiwa Lambda untuk mengakses dan sumber daya. Layanan AWS Untuk memberikan akses baca dan tulis fungsi ke bucket Amazon S3, Anda melampirkan kebijakan izin yang Anda buat di langkah sebelumnya.

Konsol Manajemen AWS

Untuk membuat peran eksekusi dan melampirkan kebijakan izin Anda (konsol)

1. Buka halaman Peran konsol (IAM).
2. Pilih Buat peran.

3. Untuk jenis entitas Tepercaya, pilih Layanan AWS, dan untuk kasus Penggunaan, pilih Lambda.
4. Pilih Berikutnya.
5. Tambahkan kebijakan izin yang Anda buat di langkah sebelumnya dengan melakukan hal berikut:
   a. Dalam kotak pencarian kebijakan, masukkan **`LambdaS3Policy`**.
   b. Dalam hasil pencarian, pilih kotak centang untuk`LambdaS3Policy`.
   c. Pilih Berikutnya.
6. Di bawah Rincian peran, untuk nama Peran masuk**`LambdaS3Role`**.
7. Pilih Buat peran.

AWS CLI

Untuk membuat peran eksekusi dan melampirkan kebijakan izin Anda (AWS CLI)

1. Simpan JSON berikut dalam file bernama`trust-policy.json`. Kebijakan kepercayaan ini memungkinkan Lambda untuk menggunakan izin peran dengan memberikan `lambda.amazonaws.com` izin utama layanan untuk memanggil tindakan AWS Security Token Service ()AWS STS. `AssumeRole`

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "lambda.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

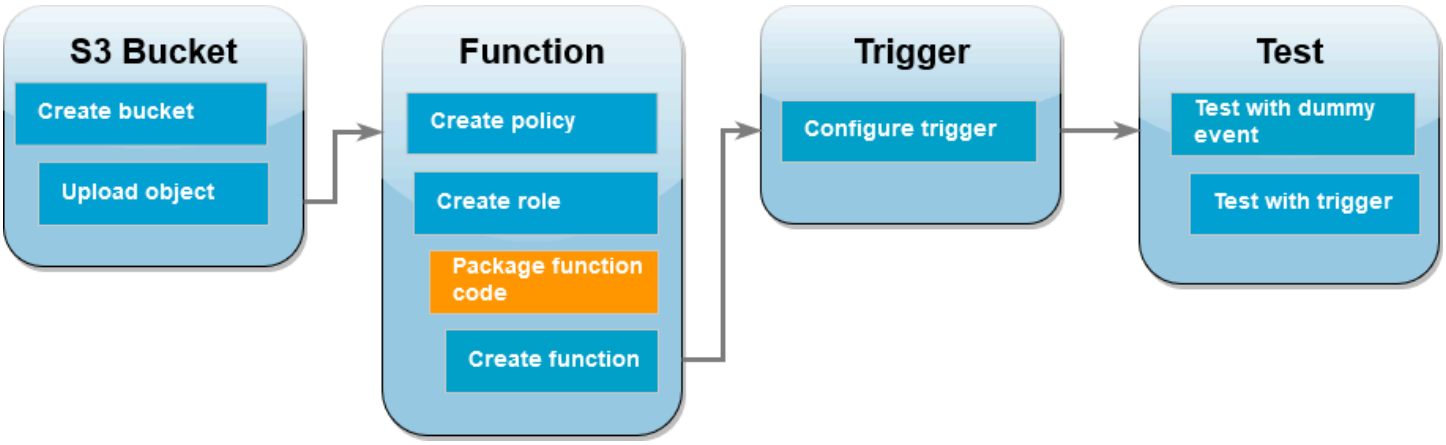
2. Dari direktori tempat Anda menyimpan dokumen kebijakan kepercayaan JSON, jalankan perintah CLI berikut untuk membuat peran eksekusi.

```
aws iam create-role --role-name LambdaS3Role --assume-role-policy-document
 file://trust-policy.json
```

3. Untuk melampirkan kebijakan izin yang Anda buat pada langkah sebelumnya, jalankan perintah CLI berikut. Ganti Akun AWS nomor di ARN polis dengan nomor akun Anda sendiri.

```
aws iam attach-role-policy --role-name LambdaS3Role --policy-arn
 arn:aws:iam::123456789012:policy/LambdaS3Policy
```

Buat paket penyebaran Fungsi Acara



Untuk membuat fungsi Anda, Anda membuat paket deployment yang berisi kode fungsi dan dependensinya. Untuk `CreateThumbnail` fungsi ini, kode fungsi Anda menggunakan pustaka terpisah untuk mengubah ukuran gambar. Ikuti instruksi untuk bahasa yang Anda pilih untuk membuat paket penyebaran yang berisi pustaka yang diperlukan.

Node.js

Untuk membuat paket penyebaran () Node.js

1. Buat direktori bernama `lambda-s3` untuk kode fungsi dan dependensi Anda dan navigasikan ke dalamnya.

```
mkdir lambda-s3
cd lambda-s3
```

2. Buat Node.js proyek baru dengan`npm`. Untuk menerima opsi default yang disediakan dalam pengalaman interaktif, tekan`Enter`.

```
npm init
```

3. Simpan kode fungsi berikut dalam file bernama`index.mjs`. Pastikan untuk mengganti `us-east-1` dengan Wilayah AWS di mana Anda membuat ember sumber dan tujuan Anda sendiri.

```
// dependencies
import { S3Client, GetObjectCommand, PutObjectCommand } from '@aws-sdk/client-
s3';

import { Readable } from 'stream';

import sharp from 'sharp';
import util from 'util';


// create S3 client
const s3 = new S3Client({region: 'us-east-1'});

// define the handler function
export const handler = async (event, context) => {

// Read options from the event parameter and get the source bucket
console.log("Reading options from event:\n", util.inspect(event, {depth: 5}));
  const srcBucket = event.Records[0].s3.bucket.name;

// Object key may have spaces or unicode non-ASCII characters
const srcKey    = decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/
g, " "));
const dstBucket = srcBucket + "-resized";
const dstKey    = "resized-" + srcKey;

// Infer the image type from the file suffix
const typeMatch = srcKey.match(/\.([^.]*)$/);
if (!typeMatch) {
  console.log("Could not determine the image type.");
  return;
}

// Check that the image type is supported
const imageType = typeMatch[1].toLowerCase();
if (imageType != "jpg" && imageType != "png") {
```

```
  console.log(`Unsupported image type: ${imageType}`);
  return;
}

// Get the image from the source bucket. GetObjectCommand returns a stream.
try {
  const params = {
    Bucket: srcBucket,
    Key: srcKey
  };
  var response = await s3.send(new GetObjectCommand(params));
  var stream = response.Body;

// Convert stream to buffer to pass to sharp resize function.
  if (stream instanceof Readable) {
    var content_buffer = Buffer.concat(await stream.toArray());

  } else {
    throw new Error('Unknown object stream type');
  }


} catch (error) {
  console.log(error);
  return;
}


// set thumbnail width. Resize will set the height automatically to maintain
 aspect ratio.
const width  = 200;

// Use the sharp module to resize the image and save in a buffer.
try {
  var output_buffer = await sharp(content_buffer).resize(width).toBuffer();

} catch (error) {
  console.log(error);
  return;
}

// Upload the thumbnail image to the destination bucket
try {
  const destparams = {
```

```
    Bucket: dstBucket,
    Key: dstKey,
    Body: output_buffer,
    ContentType: "image"
  };

  const putResult = await s3.send(new PutObjectCommand(destparams));

  } catch (error) {
    console.log(error);
    return;
  }

  console.log('Successfully resized ' + srcBucket + '/' + srcKey +
    ' and uploaded to ' + dstBucket + '/' + dstKey);
  };
```

4. Di `lambda-s3` direktori Anda, instal perpustakaan tajam menggunakan npm. Perhatikan bahwa versi terbaru dari sharp (0.33) tidak kompatibel dengan Lambda. Instal versi 0.32.6 untuk menyelesaikan tutorial ini.

```
npm install sharp@0.32.6
```

`install`Perintah npm membuat `node_modules` direktori untuk modul Anda. Setelah langkah ini, struktur direktori Anda akan terlihat seperti berikut.

```
lambda-s3
|- index.mjs
|- node_modules
|  |- base64js
|  |- bl
|  |- buffer
...
|- package-lock.json
|- package.json
```

5. Buat paket deployment .zip yang berisi kode fungsi Anda dan dependensinya. Di macOS dan Linux, jalankan perintah berikut.

```
zip -r function.zip .
```

Di Windows, gunakan utilitas zip pilihan Anda untuk membuat file.zip. Pastikan bahwa `package-lock.json` file `index.mjspackage.json`, dan `node_modules` direktori Anda semuanya berada di root file.zip Anda.

Python

Untuk membuat paket penyebaran (Python)

1. Simpan kode contoh sebagai file bernama `lambda_function.py`.

```
import boto3
import os
import sys
import uuid
from urllib.parse import unquote_plus
from PIL import Image
import PIL.Image

s3_client = boto3.client('s3')

def resize_image(image_path, resized_path):
  with Image.open(image_path) as image:
    image.thumbnail(tuple(x / 2 for x in image.size))
    image.save(resized_path)

def lambda_handler(event, context):
  for record in event['Records']:
    bucket = record['s3']['bucket']['name']
    key = unquote_plus(record['s3']['object']['key'])
    tmpkey = key.replace('/', '')
    download_path = '/tmp/{}{}'.format(uuid.uuid4(), tmpkey)
    upload_path = '/tmp/resized-{}'.format(tmpkey)
    s3_client.download_file(bucket, key, download_path)
    resize_image(download_path, upload_path)
    s3_client.upload_file(upload_path, '{}-resized'.format(bucket), 'resized-
{}'.format(key))
```

2. Di direktori yang sama di mana Anda membuat `lambda_function.py` file Anda, buat direktori baru bernama `package` dan instal pustaka Pillow (PIL) dan AWS SDK untuk Python (Boto3). Meskipun runtime Lambda Python menyertakan versi Boto3 SDK, kami menyarankan agar Anda menambahkan semua dependensi fungsi Anda ke paket penerapan

Anda, meskipun mereka disertakan dalam runtime. Untuk informasi selengkapnya, lihat Dependensi runtime dengan Python.

```
mkdir package
pip install \
--platform manylinux2014_x86_64 \
--target=package \
--implementation cp \
--python-version 3.12 \
--only-binary=:all: --upgrade \
pillow boto3
```

Pustaka Pillow berisi kode C/C ++. Dengan menggunakan `--only-binary=:all:` opsi `--platform manylinux_2014_x86_64` dan, pip mengunduh dan menginstal versi Pillow yang berisi binari pra-kompilasi yang kompatibel dengan sistem operasi Amazon Linux 2. Ini memastikan bahwa paket penerapan Anda berfungsi di lingkungan eksekusi Lambda, terlepas dari sistem operasi dan arsitektur mesin build lokal Anda.

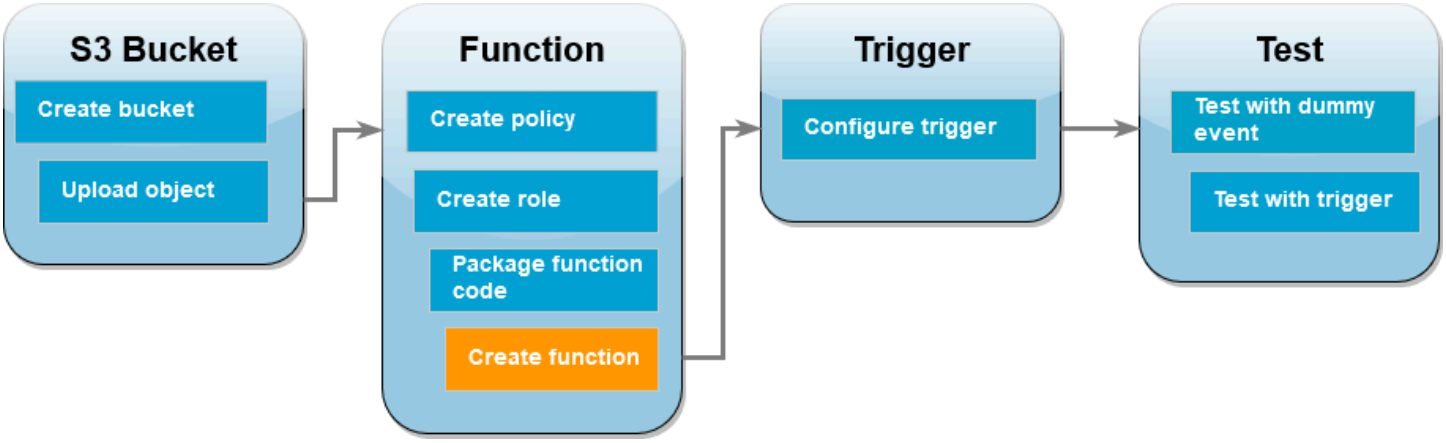
3. Buat file.zip yang berisi kode aplikasi Anda dan pustaka Pillow dan Boto3. Di Linux atau macOS, jalankan perintah berikut dari antarmuka baris perintah Anda.

```
cd package
zip -r ../lambda_function.zip .
cd ..
zip lambda_function.zip lambda_function.py
```

Di Windows, gunakan alat zip pilihan Anda untuk membuat file `lambda_function.zip`. Pastikan bahwa `lambda_function.py` file Anda dan folder yang berisi dependensi Anda semuanya berada di root file.zip.

Anda juga dapat membuat paket deployment menggunakan lingkungan virtual Python. Lihat Bekerja dengan arsip file.zip untuk Fungsi Acara Python Lambda

Buat Fungsi Acara Lambda



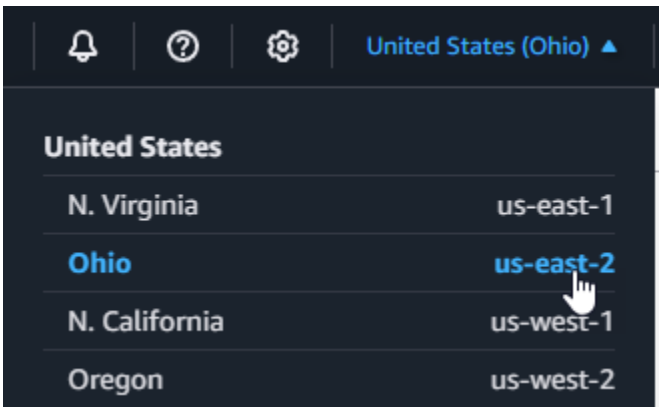
Anda dapat membuat Fungsi Acara Lambda menggunakan konsol Lambda AWS CLI atau Lambda. Ikuti instruksi untuk bahasa pilihan Anda untuk membuat fungsi.

Konsol Manajemen AWS

Untuk membuat Fungsi Acara (konsol)

Untuk membuat Fungsi Acara Lambda menggunakan konsol, pertama-tama Anda membuat fungsi dasar yang berisi beberapa kode 'Hello world'. Anda kemudian mengganti kode ini dengan kode fungsi Anda sendiri dengan mengunggah file the.zip atau JAR yang Anda buat pada langkah sebelumnya.

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pastikan Anda bekerja sama dengan saat Wilayah AWS Anda membuat bucket Amazon S3. Anda dapat mengubah wilayah Anda menggunakan daftar drop-down di bagian atas layar.



3. Pilih Buat Fungsi Acara.
4. Pilih Tulis dari awal.

5. Di bagian Informasi dasar, lakukan hal berikut:
   a. Untuk Nama fungsi, masukkan **`CreateThumbnail`**.
   b. Untuk Runtime, pilih Node.js 22.x atau Python 3.12 sesuai dengan bahasa yang Anda pilih untuk fungsi Anda.
   c. Untuk Arsitektur, pilih x86_64.
6. Di tab Ubah peran eksekusi default, lakukan hal berikut:
   a. Perluas tab, lalu pilih Gunakan peran yang ada.
   b. Pilih yang `LambdaS3Role` Anda buat sebelumnya.
7. Pilih Buat Fungsi Acara.

Untuk mengunggah kode Fungsi Acara (konsol)

1. Di panel Sumber kode, pilih Unggah dari.
2. Pilih file.zip.
3. Pilih Unggah.
4. Di pemilih file, pilih file.zip Anda dan pilih Buka.
5. Pilih Simpan.

AWS CLI

Untuk membuat Fungsi Acara (AWS CLI)

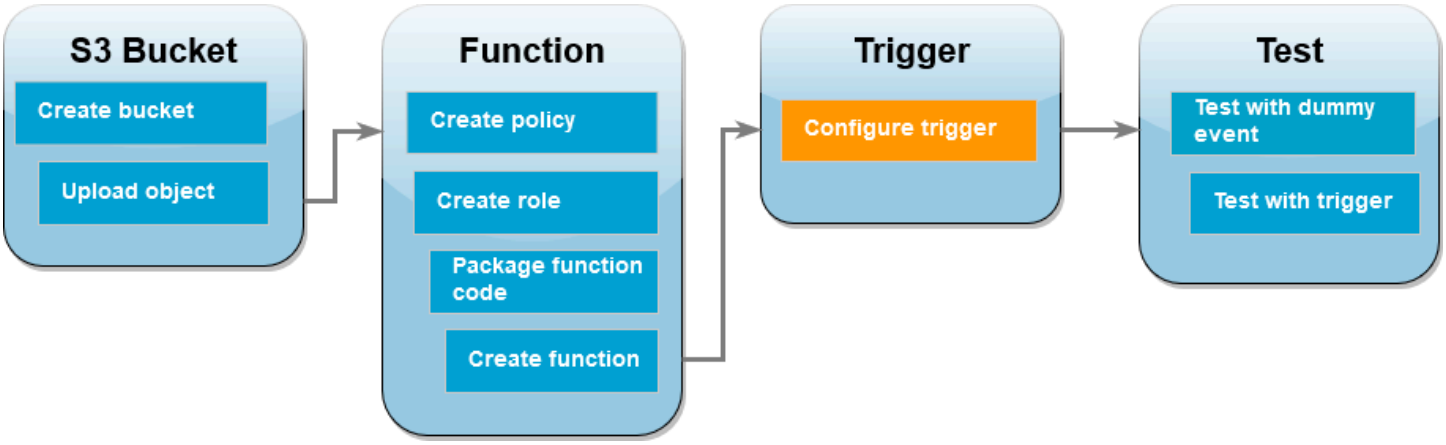
- Jalankan perintah CLI untuk bahasa yang Anda pilih. Untuk `role` parameter, pastikan untuk mengganti `123456789012` dengan Akun AWS ID Anda sendiri. Untuk `region` parameter, ganti `us-east-1` dengan wilayah tempat Anda membuat bucket Amazon S3.
  - Untuk Node.js, jalankan perintah berikut dari direktori yang berisi `function.zip` file Anda.

```
aws lambda create-function --function-name CreateThumbnail \
--zip-file fileb://function.zip --handler index.handler --runtime nodejs24.x \
--timeout 10 --memory-size 1024 \
--role arn:aws:iam::123456789012:role/LambdaS3Role --region us-east-1
```

- Untuk Python, jalankan perintah berikut dari direktori yang berisi file Andalambda_function.zip.

```
aws lambda create-function --function-name CreateThumbnail \
--zip-file fileb://lambda_function.zip --handler
 lambda_function.lambda_handler \
--runtime python3.14 --timeout 10 --memory-size 1024 \
--role arn:aws:iam::123456789012:role/LambdaS3Role --region us-east-1
```

Konfigurasikan Amazon S3 untuk menjalankan Fungsi Acara



Agar Fungsi Peristiwa Lambda dapat berjalan saat mengunggah gambar ke bucket sumber, Anda perlu mengonfigurasi pemicu untuk fungsi Anda. Anda dapat mengonfigurasi pemicu Amazon S3 menggunakan konsol atau. AWS CLI

> **Important**
>
> Prosedur ini mengonfigurasi bucket Amazon S3 untuk menjalankan fungsi Anda setiap kali objek dibuat di bucket. Pastikan untuk mengonfigurasi ini hanya di bucket sumber. Jika Fungsi Peristiwa Lambda Anda membuat objek dalam bucket yang sama yang memanggilnya, fungsi Anda dapat dipanggil terus menerus dalam satu loop. Hal ini dapat mengakibatkan biaya yang tidak diharapkan ditagih ke Anda Akun AWS.

Konsol Manajemen AWS

Untuk mengonfigurasi pemicu Amazon S3 (konsol)

1. Buka halaman Fungsi Acara konsol Lambda dan pilih fungsi Anda ()`CreateThumbnail`.
2. Pilih Tambahkan pemicu.
3. Pilih S3.
4. Di bawah Bucket, pilih bucket sumber Anda.
5. Di bawah Jenis acara, pilih Semua objek membuat acara.
6. Di bawah Pemanggilan rekursif, pilih kotak centang untuk mengetahui bahwa tidak disarankan menggunakan bucket Amazon S3 yang sama untuk input dan output. Anda dapat mempelajari lebih lanjut tentang pola pemanggilan rekursif di Lambda dengan membaca pola rekursif yang menyebabkan fungsi Lambda yang tidak terkendali di Tanah Tanpa Server.
7. Pilih Tambahkan.

   Saat Anda membuat pemicu menggunakan konsol Lambda, Lambda secara otomatis membuat kebijakan berbasis sumber daya untuk memberikan layanan yang Anda pilih izin untuk menjalankan fungsi Anda.

AWS CLI

Untuk mengonfigurasi pemicu Amazon S3 (AWS CLI)

1. Agar bucket sumber Amazon S3 menjalankan fungsi saat menambahkan file gambar, pertama-tama Anda harus mengonfigurasi izin untuk fungsi menggunakan kebijakan berbasis sumber daya. Pernyataan kebijakan berbasis sumber daya memberikan Layanan AWS izin lain untuk menjalankan fungsi Anda. Untuk memberikan izin Amazon S3 untuk menjalankan fungsi Anda, jalankan perintah CLI berikut. Pastikan untuk mengganti `source-account` parameter dengan Akun AWS ID Anda sendiri dan menggunakan nama bucket sumber Anda sendiri.

   ```
   aws lambda add-permission --function-name CreateThumbnail \
   --principal s3.amazonaws.com --statement-id s3invoke --action
    "lambda:InvokeFunction" \
   --source-arn arn:aws:s3:::amzn-s3-demo-source-bucket \
   --source-account 123456789012
   ```

Kebijakan yang Anda tetapkan dengan perintah ini memungkinkan Amazon S3 untuk menjalankan fungsi Anda hanya ketika tindakan dilakukan di bucket sumber Anda.

> **Note**
>
> Meskipun nama bucket Amazon S3 unik secara global, saat menggunakan kebijakan berbasis sumber daya, praktik terbaik adalah menentukan bahwa bucket harus menjadi milik akun Anda. Ini karena jika Anda menghapus bucket, Anda dapat membuat bucket dengan Amazon Resource Name (ARN) yang sama. Akun AWS

2. Simpan JSON berikut dalam file bernama`notification.json`. Saat diterapkan ke bucket sumber Anda, JSON ini mengonfigurasi bucket untuk mengirim notifikasi ke Fungsi Peristiwa Lambda Anda setiap kali objek baru ditambahkan. Ganti Akun AWS nomor dan Wilayah AWS di ARN Fungsi Acara Lambda dengan nomor akun dan wilayah Anda sendiri.

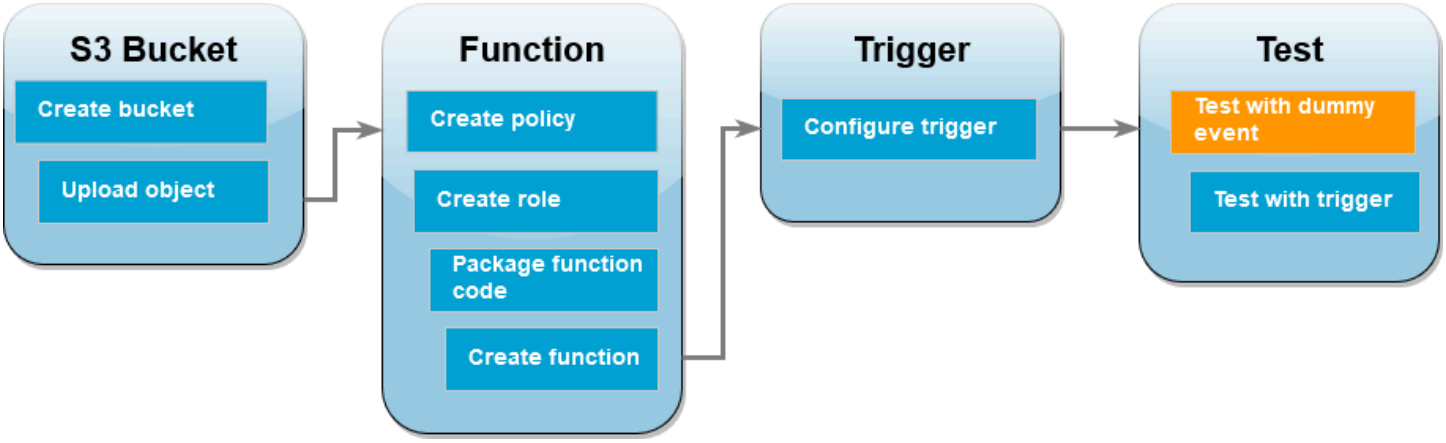
```
{
"LambdaFunctionConfigurations": [
    {
      "Id": "CreateThumbnailEventConfiguration",
      "LambdaFunctionArn": "arn:aws:lambda:us-
east-1:123456789012:function:CreateThumbnail",
      "Events": [ "s3:ObjectCreated:Put" ]
    }
  ]
}
```

3. Jalankan perintah CLI berikut untuk menerapkan pengaturan notifikasi dalam file JSON yang Anda buat ke bucket sumber Anda. Ganti `amzn-s3-demo-source-bucket` dengan nama bucket sumber Anda sendiri.

```
aws s3api put-bucket-notification-configuration --bucket amzn-s3-demo-source-
bucket \
--notification-configuration file://notification.json
```

Untuk mempelajari lebih lanjut tentang *`put-bucket-notification-configuration`* perintah dan *`notification-configuration`* opsi, lihat put-bucket-notification-configuration di CLI Command Reference.AWS

Uji Fungsi Acara Lambda Anda dengan acara dummy



Sebelum menguji seluruh penyiapan dengan menambahkan file gambar ke bucket sumber Amazon S3, Anda menguji apakah Fungsi Peristiwa Lambda berfungsi dengan benar dengan memanggilnya dengan acara dummy. Peristiwa di Lambda adalah JSON-formatted dokumen yang berisi data untuk fungsi Anda untuk diproses. Saat fungsi Anda dipanggil oleh Amazon S3, peristiwa yang dikirim ke fungsi berisi informasi seperti nama bucket, ARN bucket, dan kunci objek.

Konsol Manajemen AWS

Untuk menguji Fungsi Acara Lambda Anda dengan acara dummy (konsol)

1. Buka halaman Fungsi Acara konsol Lambda dan pilih fungsi Anda ()`CreateThumbnail`.
2. Pilih tab Uji.
3. Untuk membuat acara pengujian, di panel acara Uji, lakukan hal berikut:
   a. Di bawah Uji tindakan peristiwa, pilih Buat acara baru.
   b. Untuk Nama peristiwa, masukkan **`myTestEvent`**.
   c. Untuk Template, pilih S3 Put.
   d. Ganti nilai untuk parameter berikut dengan nilai Anda sendiri.
      - Untuk`awsRegion`, ganti `us-east-1` dengan bucket Amazon S3 yang Wilayah AWS Anda buat.
      - Untuk`name`, ganti `amzn-s3-demo-bucket` dengan nama bucket sumber Amazon S3 Anda sendiri.
      - Untuk`key`, ganti `test%2Fkey` dengan nama file objek pengujian yang Anda unggah ke bucket sumber di langkah tersebut. Unggah gambar uji ke bucket sumber Anda

```
{
  "Records": [
    {
      "eventVersion": "2.0",
      "eventSource": "aws:s3",
      "awsRegion": "us-east-1",
      "eventTime": "1970-01-01T00:00:00.000Z",
      "eventName": "ObjectCreated:Put",
      "userIdentity": {
        "principalId": "EXAMPLE"
      },
      "requestParameters": {
        "sourceIPAddress": "127.0.0.1"
      },
      "responseElements": {
        "x-amz-request-id": "EXAMPLE123456789",
        "x-amz-id-2": "EXAMPLE123/5678abcdefghijklambdaisawesome/
mnopqrstuvwxyzABCDEFGH"
      },
      "s3": {
        "s3SchemaVersion": "1.0",
        "configurationId": "testConfigRule",
        "bucket": {
          "name": "amzn-s3-demo-bucket",
          "ownerIdentity": {
            "principalId": "EXAMPLE"
          },
          "arn": "arn:aws:s3:::amzn-s3-demo-bucket"
        },
        "object": {
          "key": "test%2Fkey",
          "size": 1024,
          "eTag": "0123456789abcdef0123456789abcdef",
          "sequencer": "0A1B2C3D4E5F678901"
        }
      }
    }
  ]
}
```

e. Pilih Simpan.

4. Di panel acara Uji, pilih Uji.
5. Untuk memeriksa fungsi Anda telah membuat verison yang diubah ukurannya dari gambar Anda dan menyimpannya di bucket Amazon S3 target Anda, lakukan hal berikut:

   a. Buka halaman Bucket konsol Amazon S3.

   b. Pilih bucket target Anda dan konfirmasikan bahwa file yang diubah ukurannya tercantum di panel Objects.

AWS CLI

Untuk menguji Fungsi Acara Lambda Anda dengan acara dummy (AWS CLI)

1. Simpan JSON berikut dalam file bernama`dummyS3Event.json`. Ganti nilai untuk parameter berikut dengan nilai Anda sendiri:

   - Untuk`awsRegion`, ganti `us-east-1` dengan bucket Amazon S3 yang Wilayah AWS Anda buat.
   - Untuk`name`, ganti `amzn-s3-demo-bucket` dengan nama bucket sumber Amazon S3 Anda sendiri.
   - Untuk`key`, ganti `test%2Fkey` dengan nama file objek pengujian yang Anda unggah ke bucket sumber di langkah tersebut. Unggah gambar uji ke bucket sumber Anda

```
{
  "Records": [
    {
      "eventVersion": "2.0",
      "eventSource": "aws:s3",
      "awsRegion": "us-east-1",
      "eventTime": "1970-01-01T00:00:00.000Z",
      "eventName": "ObjectCreated:Put",
      "userIdentity": {
        "principalId": "EXAMPLE"
      },
      "requestParameters": {
        "sourceIPAddress": "127.0.0.1"
      },
      "responseElements": {
        "x-amz-request-id": "EXAMPLE123456789",
```

```
        "x-amz-id-2": "EXAMPLE123/5678abcdefghijklambdaisawesome/
mnopqrstuvwxyzABCDEFGH"
      },
      "s3": {
        "s3SchemaVersion": "1.0",
        "configurationId": "testConfigRule",
        "bucket": {
          "name": "amzn-s3-demo-bucket",
          "ownerIdentity": {
            "principalId": "EXAMPLE"
          },
          "arn": "arn:aws:s3:::amzn-s3-demo-bucket"
        },
        "object": {
          "key": "test%2Fkey",
          "size": 1024,
          "eTag": "0123456789abcdef0123456789abcdef",
          "sequencer": "0A1B2C3D4E5F678901"
        }
      }
    }
  ]
}
```

2. Dari direktori tempat Anda menyimpan `dummyS3Event.json` file Anda, panggil fungsi dengan menjalankan perintah CLI berikut. Perintah ini memanggil Fungsi Peristiwa Lambda Anda secara sinkron dengan `RequestResponse` menentukan sebagai nilai parameter tipe pemanggilan. Untuk mempelajari lebih lanjut tentang pemanggilan sinkron dan asinkron, lihat Memanggil fungsi Lambda.

```
aws lambda invoke --function-name CreateThumbnail \
--invocation-type RequestResponse --cli-binary-format raw-in-base64-out \
--payload file://dummyS3Event.json outputfile.txt
```

   Opsi format cli-biner diperlukan jika Anda menggunakan versi 2. AWS CLI Untuk menjadikan ini pengaturan default, jalankan`aws configure set cli-binary-format raw-in-base64-out`. Untuk informasi selengkapnya, lihat opsi baris perintah global yang AWS CLI didukung.

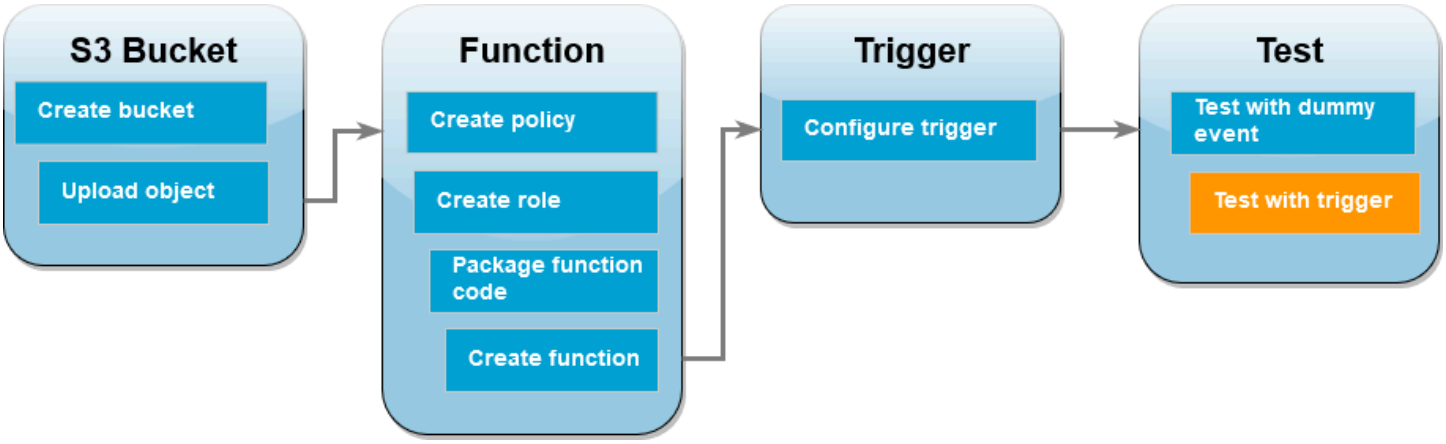
3. Verifikasi bahwa fungsi Anda telah membuat versi thumbnail gambar Anda dan menyimpannya ke bucket Amazon S3 target Anda. Jalankan perintah CLI berikut, ganti `amzn-s3-demo-source-bucket-resized` dengan nama bucket tujuan Anda sendiri.

```
aws s3api list-objects-v2 --bucket amzn-s3-demo-source-bucket-resized
```

Anda akan melihat output seperti yang berikut ini. KeyParameter menunjukkan nama file file gambar Anda yang diubah ukurannya.

```
{
    "Contents": [
        {
            "Key": "resized-HappyFace.jpg",
            "LastModified": "2023-06-06T21:40:07+00:00",
            "ETag": "\"d8ca652ffe83ba6b721ffc20d9d7174a\"",
            "Size": 2633,
            "StorageClass": "STANDARD"
        }
    ]
}
```

Uji Fungsi Acara Anda menggunakan pemicu Amazon S3



Setelah mengonfirmasi bahwa Fungsi Acara Lambda Anda beroperasi dengan benar, Anda siap untuk menguji penyiapan lengkap Anda dengan menambahkan file gambar ke bucket sumber Amazon S3 Anda. Saat Anda menambahkan gambar ke bucket sumber, Fungsi Peristiwa Lambda Anda akan dipanggil secara otomatis. Fungsi Anda membuat versi file yang diubah ukurannya dan menyimpannya di bucket target Anda.

Konsol Manajemen AWS

Untuk menguji Fungsi Peristiwa Lambda menggunakan pemicu Amazon S3 (konsol)

1. Untuk mengunggah gambar ke bucket Amazon S3 Anda, lakukan hal berikut:
   a. Buka halaman Bucket di konsol Amazon S3 dan pilih bucket sumber Anda.
   b. Pilih Unggah.
   c. Pilih Tambahkan file dan gunakan pemilih file untuk memilih file gambar yang ingin Anda unggah. Objek gambar Anda dapat berupa file.jpg atau.png.
   d. Pilih Buka, lalu pilih Unggah.
2. Verifikasi bahwa Lambda telah menyimpan versi file gambar yang diubah ukurannya di bucket target dengan melakukan hal berikut:
   a. Arahkan kembali ke halaman Bucket di konsol Amazon S3 dan pilih bucket tujuan Anda.
   b. Di panel Objects, Anda sekarang akan melihat dua file gambar yang diubah ukurannya, satu dari setiap pengujian Fungsi Peristiwa Lambda Anda. Untuk mengunduh gambar yang diubah ukurannya, pilih file, lalu pilih Unduh.

AWS CLI

Untuk menguji Fungsi Peristiwa Lambda Anda menggunakan pemicu Amazon S3 (AWS CLI)

1. Dari direktori yang berisi gambar yang ingin Anda unggah, jalankan perintah CLI berikut. Ganti `--bucket` parameter dengan nama bucket sumber Anda. Untuk `--body` parameter `--key` dan, gunakan nama file gambar pengujian Anda. Gambar uji Anda dapat berupa file.jpg atau.png.

```
aws s3api put-object --bucket amzn-s3-demo-source-bucket --key SmileyFace.jpg --body ./SmileyFace.jpg
```

2. Verifikasi bahwa fungsi Anda telah membuat versi thumbnail gambar Anda dan menyimpannya ke bucket Amazon S3 target Anda. Jalankan perintah CLI berikut, ganti `amzn-s3-demo-source-bucket-resized` dengan nama bucket tujuan Anda sendiri.

```
aws s3api list-objects-v2 --bucket amzn-s3-demo-source-bucket-resized
```

Jika fungsi Anda berjalan dengan sukses, Anda akan melihat output yang mirip dengan berikut ini. Bucket target Anda sekarang harus berisi dua file yang diubah ukurannya.


```
{
    "Contents": [
        {
            "Key": "resized-HappyFace.jpg",
            "LastModified": "2023-06-07T00:15:50+00:00",
            "ETag": "\"7781a43e765a8301713f533d70968a1e\"",
            "Size": 2763,
            "StorageClass": "STANDARD"
        },
        {
            "Key": "resized-SmileyFace.jpg",
            "LastModified": "2023-06-07T00:13:18+00:00",
            "ETag": "\"ca536e5a1b9e32b22cd549e18792cdbc\"",
            "Size": 1245,
            "StorageClass": "STANDARD"
        }
    ]
}
```


Bersihkan sumber daya Anda

Sekarang Anda dapat menghapus sumber daya yang Anda buat untuk tutorial ini, kecuali Anda ingin mempertahankannya. Dengan menghapus AWS sumber daya yang tidak lagi Anda gunakan, Anda mencegah tagihan yang tidak perlu ke Anda Akun AWS.

Untuk menghapus Fungsi Acara Lambda

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih Fungsi Acara yang Anda buat.
3. Pilih Tindakan, Hapus.
4. Ketik **`confirm`** kolom input teks dan pilih Hapus.

Untuk menghapus kebijakan yang Anda buat.

1. Buka halaman Kebijakan konsol IAM.

2. Pilih kebijakan yang Anda buat (AWSLambdaS3Policy).
3. Pilih Tindakan kebijakan, Hapus.
4. Pilih Hapus.

Untuk menghapus peran eksekusi

1. Buka halaman Peran dari konsol IAM.
2. Pilih peran eksekusi yang Anda buat.
3. Pilih Hapus.
4. Masukkan nama peran di bidang input teks dan pilih Hapus.

Untuk menghapus bucket S3

1. Buka konsol Amazon S3.
2. Pilih bucket yang Anda buat.
3. Pilih Hapus.
4. Masukkan nama ember di bidang input teks.
5. Pilih Hapus bucket.

Gunakan rahasia Secrets Manager di Lambda Event Functions

AWS Secrets Manager membantu Anda mengelola kredensi, kunci API, dan rahasia lain yang dibutuhkan Fungsi Peristiwa Lambda Anda. Anda memiliki dua pendekatan utama untuk mengambil rahasia di Fungsi Peristiwa Lambda Anda, keduanya menawarkan kinerja yang lebih baik dan biaya yang lebih rendah dibandingkan dengan mengambil rahasia secara langsung menggunakan SDK: AWS

- AWS parameter dan rahasia Ekstensi Lambda - Solusi runtime-agnostik yang menyediakan antarmuka HTTP sederhana untuk mengambil rahasia
- Powertools untuk utilitas AWS Lambda parameter - Solusi terintegrasi kode yang mendukung beberapa penyedia (Secrets Manager, Parameter Store, AppConfig) dengan transformasi bawaan

Kedua pendekatan mempertahankan cache rahasia lokal, menghilangkan kebutuhan fungsi Anda untuk memanggil Secrets Manager untuk setiap pemanggilan. Ketika fungsi Anda meminta rahasia, cache diperiksa terlebih dahulu. Jika rahasia tersedia dan belum kedaluwarsa, itu segera dikembalikan. Jika tidak, itu diambil dari Secrets Manager, di-cache, dan dikembalikan. Mekanisme caching ini menghasilkan waktu respons yang lebih cepat dan mengurangi biaya dengan meminimalkan panggilan API.

Memilih pendekatan

Pertimbangkan faktor-faktor ini ketika memilih antara ekstensi dan PowerTools:

Gunakan AWS parameter dan rahasia ekstensi Lambda saat:

- Anda menginginkan solusi runtime-agnostik yang berfungsi dengan runtime Lambda apa pun
- Anda memilih untuk tidak menambahkan dependensi kode ke fungsi Anda
- Anda hanya perlu mengambil rahasia dari Secrets Manager atau Parameter Store

Gunakan Powertools untuk utilitas AWS Lambda parameter saat:

- Anda menginginkan pengalaman pengembangan terintegrasi dengan kode aplikasi Anda
- Anda memerlukan dukungan untuk beberapa penyedia (Secrets Manager, Parameter Store, AppConfig)
- Anda ingin transformasi data bawaan (penguraian JSON, decoding base64)
- Anda menggunakan runtime Python,, Java TypeScript, atau .NET

Kapan menggunakan Secrets Manager dengan Lambda

Skenario umum untuk menggunakan Secrets Manager dengan Lambda meliputi:

- Menyimpan kredensi database yang digunakan fungsi Anda untuk terhubung ke Amazon RDS atau database lainnya
- Mengelola kunci API untuk layanan eksternal yang dipanggil fungsi Anda
- Menyimpan kunci enkripsi atau data konfigurasi sensitif lainnya
- Memutar kredensil secara otomatis tanpa perlu memperbarui kode fungsi Anda

Menggunakan AWS parameter dan rahasia ekstensi Lambda

AWS Parameter dan rahasia ekstensi Lambda menggunakan antarmuka HTTP sederhana yang kompatibel dengan runtime Lambda apa pun. Secara default, ini menyimpan rahasia selama 300 detik (5 menit) dan dapat menampung hingga 1.000 rahasia. Anda dapat menyesuaikan pengaturan ini dengan variabel lingkungan.

Gunakan Secrets Manager dalam Fungsi Acara Lambda

Bagian ini mengasumsikan bahwa Anda sudah memiliki rahasia Secrets Manager. Untuk membuat rahasia, lihat Membuat AWS Secrets Manager rahasia.

Buat paket deployment

Pilih runtime pilihan Anda dan ikuti langkah-langkah untuk membuat fungsi yang mengambil rahasia dari Secrets Manager. Fungsi contoh mengambil rahasia dari Secrets Manager dan dapat digunakan untuk mengakses kredensi database, kunci API, atau data konfigurasi sensitif lainnya dalam aplikasi Anda.

Python

Untuk membuat Fungsi Acara Python

1. Buat dan arahkan ke direktori proyek baru. Contoh:

```
mkdir my_function
cd my_function
```

2. Buat file bernama `lambda_function.py` dengan kode berikut. Untuk`secret_name`, gunakan nama atau Amazon Resource Name (ARN) rahasia Anda.

```
import json
import boto3
import requests

def lambda_handler(event, context):
    try:
        # Replace with the name or ARN of your secret
        secret_name = "arn:aws:secretsmanager:us-
east-1:111122223333:secret:SECRET_NAME"

        # Resolve the session token from the credential provider chain so that
 this
        # works across all initialization modes, including SnapStart. Resolve
 inside
        # the handler so the token stays fresh after a SnapStart restore.
        session_token =
 boto3.Session().get_credentials().get_frozen_credentials().token

        secrets_extension_endpoint = f"http://localhost:2773/secretsmanager/get?
secretId={secret_name}"
        headers = {"X-Aws-Parameters-Secrets-Token": session_token}

        response = requests.get(secrets_extension_endpoint, headers=headers)
        print(f"Response status code: {response.status_code}")

        secret = json.loads(response.text)["SecretString"]
        print(f"Retrieved secret: {secret}")

        return {
            'statusCode': response.status_code,
            'body': json.dumps({
                'message': 'Successfully retrieved secret',
                'secretRetrieved': True
            })
        }

    except Exception as e:
        print(f"Error: {str(e)}")
        return {
            'statusCode': 500,
            'body': json.dumps({
                'message': 'Error retrieving secret',
                'error': str(e)
```

```
            })
        }
```

3. Buat file bernama `requirements.txt` dengan konten ini:

```
requests
```

4. Instal dependensi:

```
pip install -r requirements.txt -t .
```

5. Buat file.zip yang berisi semua file:

```
zip -r function.zip .
```

Node.js

Untuk membuat Fungsi Node.js Acara

1. Buat dan arahkan ke direktori proyek baru. Contoh:

```
mkdir my_function
cd my_function
```

2. Buat file bernama `index.mjs` dengan kode berikut. Untuk`secret_name`, gunakan nama atau Amazon Resource Name (ARN) rahasia Anda.

```
import http from 'http';
import { fromNodeProviderChain } from '@aws-sdk/credential-providers';

export const handler = async (event) => {
    try {
        // Replace with the name or ARN of your secret
        const secretName = "arn:aws:secretsmanager:us-
east-1:111122223333:secret:SECRET_NAME";

        // Resolve the session token from the credential provider chain so that
 this
        // works across all initialization modes, including SnapStart. Resolve
 inside
        // the handler so the token stays fresh after a SnapStart restore.
```

```
        const { sessionToken } = await fromNodeProviderChain()();

        const options = {
            hostname: 'localhost',
            port: 2773,
            path: `/secretsmanager/get?secretId=${secretName}`,
            headers: {
                'X-Aws-Parameters-Secrets-Token': sessionToken
            }
        };

        const response = await new Promise((resolve, reject) => {
            http.get(options, (res) => {
                let data = '';
                res.on('data', (chunk) => { data += chunk; });
                res.on('end', () => {
                    resolve({
                        statusCode: res.statusCode,
                        body: data
                    });
                });
            }).on('error', reject);
        });

        const secret = JSON.parse(response.body).SecretString;
        console.log('Retrieved secret:', secret);

        return {
            statusCode: response.statusCode,
            body: JSON.stringify({
                message: 'Successfully retrieved secret',
                secretRetrieved: true
            })
        };
    } catch (error) {
        console.error('Error:', error);
        return {
            statusCode: 500,
            body: JSON.stringify({
                message: 'Error retrieving secret',
                error: error.message
            })
        };
    }
```

```
};
```

3. Buat file.zip yang berisi `index.mjs` file:

```
zip -r function.zip index.mjs
```

Java

Untuk membuat Java Event Function

1. Buat proyek Maven:

```
mvn archetype:generate \
    -DgroupId=example \
    -DartifactId=lambda-secrets-demo \
    -DarchetypeArtifactId=maven-archetype-quickstart \
    -DarchetypeVersion=1.4 \
    -DinteractiveMode=false
```

2. Arahkan ke direktori proyek:

```
cd lambda-secrets-demo
```

3. Buka `pom.xml` dan ganti isinya dengan yang berikut:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>example</groupId>
    <artifactId>lambda-secrets-demo</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>11</maven.compiler.source>
        <maven.compiler.target>11</maven.compiler.target>
    </properties>

    <dependencies>
        <dependency>
```

```
            <groupId>com.amazonaws</groupId>
            <artifactId>aws-lambda-java-core</artifactId>
            <version>1.2.1</version>
        </dependency>
        <dependency>
            <groupId>software.amazon.awssdk</groupId>
            <artifactId>auth</artifactId>
            <version>2.25.0</version>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-shade-plugin</artifactId>
                <version>3.2.4</version>
                <executions>
                    <execution>
                        <phase>package</phase>
                        <goals>
                            <goal>shade</goal>
                        </goals>
                        <configuration>
                            <createDependencyReducedPom>false</
createDependencyReducedPom>
                            <finalName>function</finalName>
                        </configuration>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
</project>
```

4. Ganti nama `/lambda-secrets-demo/src/main/java/example/App.java` `Hello.java` untuk mencocokkan nama penangan Java default Lambda (): `example.Hello::handleRequest`

```
mv src/main/java/example/App.java src/main/java/example/Hello.java
```

5. Buka `Hello.java` file dan ganti isinya dengan yang berikut ini. Untuk`secretName`, gunakan nama atau Amazon Resource Name (ARN) rahasia Anda.

```
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.auth.credentials.AwsSessionCredentials;
import java.net.URI;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;
import java.net.http.HttpResponse;

public class Hello implements RequestHandler<Object, String> {
    private final HttpClient client = HttpClient.newHttpClient();
    private final DefaultCredentialsProvider credentialsProvider =
 DefaultCredentialsProvider.create();

    @Override
    public String handleRequest(Object input, Context context) {
        try {
            // Replace with the name or ARN of your secret
            String secretName = "arn:aws:secretsmanager:us-
east-1:111122223333:secret:SECRET_NAME";
            String endpoint = "http://localhost:2773/secretsmanager/get?
secretId=" + secretName;

            // Resolve the session token from the credential provider chain so
 that this
            // works across all initialization modes, including SnapStart.
 Resolve inside
            // the handler so the token stays fresh after a SnapStart restore.
            AwsSessionCredentials credentials = (AwsSessionCredentials)
 credentialsProvider.resolveCredentials();
            String sessionToken = credentials.sessionToken();

            HttpRequest request = HttpRequest.newBuilder()
                .uri(URI.create(endpoint))
                .header("X-Aws-Parameters-Secrets-Token", sessionToken)
                .GET()
                .build();

            HttpResponse<String> response = client.send(request,
                HttpResponse.BodyHandlers.ofString());
```

```
            String secret = response.body();
            secret = secret.substring(secret.indexOf("SecretString") + 15);
            secret = secret.substring(0, secret.indexOf("\""));

            System.out.println("Retrieved secret: " + secret);
            return String.format(
                "{\"statusCode\": %d, \"body\": \"%s\"}",
                response.statusCode(), "Successfully retrieved secret"
            );

        } catch (Exception e) {
            e.printStackTrace();
            return String.format(
                "{\"body\": \"Error retrieving secret: %s\"}",
                e.getMessage()
            );
        }
    }
}
```

6. Hapus direktori pengujian. Maven membuat ini secara default, tetapi kami tidak membutuhkannya untuk contoh ini.

```
rm -rf src/test
```

7. Membangun proyek:

```
mvn package
```

8. Unduh file JAR (`target/function.jar`) untuk digunakan nanti.

Buat Fungsi Acara

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih Buat Fungsi Acara.
3. Pilih Penulis dari awal.
4. Untuk Nama fungsi, masukkan **`secret-retrieval-demo`**.
5. Pilih Runtime pilihan Anda.
6. Pilih Buat Fungsi Acara.

Untuk meng-upload paket deployment

1. Di tab Kode fungsi, pilih Unggah dari dan pilih file.zip (untuk Python Node.js dan) atau file.jar (untuk Java).
2. Unggah paket penerapan yang Anda buat sebelumnya.
3. Pilih Simpan.

Tambahkan ekstensi

Untuk menambahkan AWS Parameter dan Rahasia Ekstensi Lambda sebagai lapisan

1. Di tab Code fungsi, gulir ke bawah ke Layers.
2. Pilih Tambahkan lapisan.
3. Pilih AWS layer.
4. Pilih AWS- Parameters-and-Secrets-Lambda-Extension.
5. Pilih versi terbaru.
6. Pilih Tambahkan.

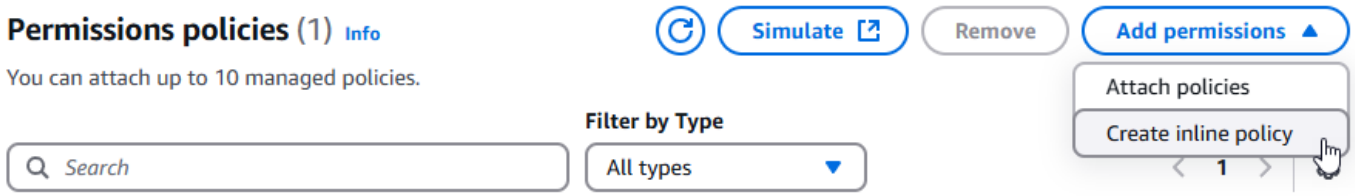
Tambahkan izin

Untuk menambahkan izin Secrets Manager ke peran eksekusi

1. Pilih tab Konfigurasi, lalu pilih Izin.
2. Di bawah Nama peran, pilih tautan ke peran eksekusi Anda. Tautan ini membuka peran di konsol IAM.



3. Pilih Tambahkan izin, lalu pilih Buat kebijakan sebaris.



4. Pilih tab JSON dan tambahkan kebijakan berikut. Untuk`Resource`, masukkan ARN rahasia Anda.

   JSON

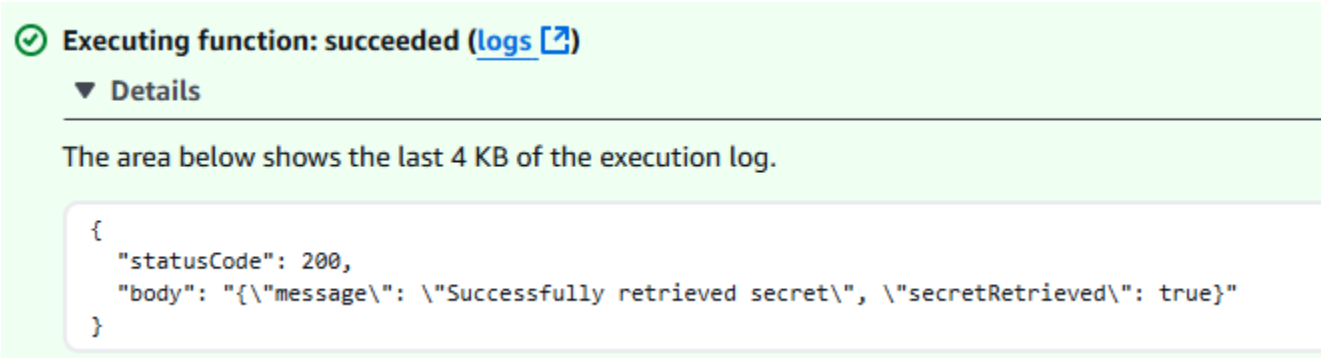
```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "secretsmanager:GetSecretValue",
            "Resource": "arn:aws:secretsmanager:us-
east-1:111122223333:secret:SECRET_NAME"
        }
    ]
}
```

5. Pilih Berikutnya.
6. Masukkan nama untuk kebijakan.
7. Pilih Buat kebijakan.

Uji Fungsi Acara

Untuk menguji Fungsi Acara

1. Kembali ke konsol Lambda.
2. Pilih tab Uji.
3. Pilih Uji. Anda akan melihat tanggapan berikut:



Variabel lingkungan

Ekstensi AWS Parameter dan Rahasia Lambda menggunakan pengaturan default berikut. Anda dapat mengganti pengaturan ini dengan membuat variabel lingkungan yang sesuai. Untuk melihat pengaturan saat ini untuk suatu fungsi, setel `PARAMETERS_SECRETS_EXTENSION_LOG_LEVEL` ke`DEBUG`. Ekstensi akan mencatat informasi konfigurasinya ke CloudWatch Log pada awal setiap pemanggilan fungsi.

Pengaturan	Nilai default	Nilai valid	Variabel lingkungan	Detail
Port HTTP	2773	1 - 65535	PARAMETER_SECRETS_EXTENSION_HTTP_PORT	Port untuk server HTTP lokal
Cache diaktifkan	BETUL	TRUE | FALSE	PARAMETERS_SECRETS_EXTENSION_CACHE_ENABLED	Aktifkan atau nonaktifkan cache
Ukuran cache	1000	0 - 1000	PARAMETER_SECRETS_EXTENSION_CACHE_SIZE	Setel ke 0 untuk menonaktifkan caching
Secrets Manager TTL	300 detik	0 - 300 detik	SECRETS_MANAGER_TTL	Time-to-live untuk rahasia yang di-cache. Setel ke 0 untuk menonaktifkan caching. Variabel ini diabaikan jika nilai untuk PARAMETERS_SECRETS_EXTENSION_CACHE_SIZE adalah 0.
Parameter Toko TTL	300 detik	0 - 300 detik	SSM_PARAMETER_STORE_TTL	Time-to-live untuk parameter cache. Setel ke 0 untuk menonaktifkan caching. Variabel ini diabaikan jika nilai

Pengaturan	Nilai default	Nilai valid	Variabel lingkungan	Detail
				untuk PARAMETERS_SECRETS_EXTENSION_CACHE_SIZE adalah 0.
Tingkat log	INFO	DEBUG | INFO | PERINGATAN | KESALAHAN | TIDAK ADA	PARAMETER_SECRETS_EXTENSION_LOG_LEVEL	Tingkat detail yang dilaporkan dalam log untuk ekstensi
Koneksi maks	3	1 atau lebih	PARAMETER_SECRETS_EXTENSION_MAX_CONNECTIONS	Jumlah maksimum koneksi HTTP untuk permintaan ke Parameter Store atau Secrets Manager
Batas waktu Secrets Manager	0 (tidak ada batas waktu)	Semua bilangan bulat	SECRETS_MANAGER_TIMEOUT_MILLIS	Batas waktu untuk permintaan ke Secrets Manager (dalam milidetik)
Batas waktu penyimpanan Parameter	0 (tidak ada batas waktu)	Semua bilangan bulat	SSM_PARAMETER_STORE_TIMEOUT_MILLIS	Batas waktu untuk permintaan ke Parameter Store (dalam milidetik)

Bekerja dengan rotasi rahasia

Jika Anda sering memutar rahasia, durasi cache default 300 detik dapat menyebabkan fungsi Anda menggunakan rahasia yang sudah ketinggalan zaman. Anda memiliki dua opsi untuk memastikan fungsi Anda menggunakan nilai rahasia terbaru:

- Kurangi cache TTL dengan mengatur variabel `SECRETS_MANAGER_TTL` lingkungan ke nilai yang lebih rendah (dalam hitungan detik). Misalnya, menyetelnya untuk `60` memastikan fungsi Anda tidak akan pernah menggunakan rahasia yang berumur lebih dari satu menit.
- Gunakan label `AWSCURRENT` atau `AWSPREVIOUS` pementasan dalam permintaan rahasia Anda untuk memastikan Anda mendapatkan versi spesifik yang Anda inginkan:

```
secretsmanager/get?secretId=YOUR_SECRET_NAME&versionStage=AWSCURRENT
```

Pilih pendekatan yang paling menyeimbangkan kebutuhan Anda akan kinerja dan kesegaran. TTL yang lebih rendah berarti panggilan yang lebih sering ke Secrets Manager tetapi memastikan Anda bekerja dengan nilai rahasia terbaru.

Menggunakan utilitas parameter dari Powertools untuk AWS Lambda

Utilitas parameter dari Powertools untuk AWS Lambda menyediakan antarmuka terpadu untuk mengambil rahasia dari beberapa penyedia termasuk Secrets Manager, penyimpanan parameter, dan. AppConfig Ini menangani caching, transformasi, dan memberikan pengalaman pengembangan yang lebih terintegrasi dibandingkan dengan pendekatan ekstensi.

Manfaat utilitas parameter

- Beberapa penyedia - Ambil parameter dari Secrets Manager, Parameter Store, dan AppConfig menggunakan antarmuka yang sama
- Built-in transformasi - Penguraian JSON otomatis, decoding base64, dan transformasi data lainnya
- Caching terintegrasi - Caching yang dapat dikonfigurasi dengan dukungan TTL untuk mengurangi panggilan API
- Keamanan tipe - Dukungan pengetikan yang kuat di TypeScript dan runtime lain yang didukung
- Penanganan kesalahan - Built-in coba lagi logika dan penanganan kesalahan

Contoh kode

Contoh berikut menunjukkan cara mengambil rahasia menggunakan utilitas Parameter di runtime yang berbeda:

Python

> **Note**
>
> Untuk contoh lengkap dan petunjuk penyiapan, lihat dokumentasi utilitas Parameter.

Mengambil rahasia dari Secrets Manager dengan Powertools untuk utilitas AWS Lambda Parameter.

```
from aws_lambda_powertools import Logger
from aws_lambda_powertools.utilities import parameters

logger = Logger()

def lambda_handler(event, context):
    try:
        # Get secret with caching (default TTL: 5 seconds)
        secret_value = parameters.get_secret("my-secret-name")

        # Get secret with custom TTL
        secret_with_ttl = parameters.get_secret("my-secret-name", max_age=300)

        # Get secret and transform JSON
        secret_json = parameters.get_secret("my-json-secret", transform="json")

        logger.info("Successfully retrieved secrets")

        return {
            'statusCode': 200,
            'body': 'Successfully retrieved secrets'
        }

    except Exception as e:
        logger.error(f"Error retrieving secret: {str(e)}")
        return {
            'statusCode': 500,
            'body': f'Error: {str(e)}'
        }
```

TypeScript

> **Note**
>
> Untuk contoh lengkap dan petunjuk penyiapan, lihat dokumentasi utilitas Parameter.

Mengambil rahasia dari Secrets Manager dengan Powertools untuk utilitas AWS Lambda Parameter.

```
import { Logger } from '@aws-lambda-powertools/logger';
import { getSecret } from '@aws-lambda-powertools/parameters/secrets';
import type { Context } from 'aws-lambda';

const logger = new Logger();

export const handler = async (event: any, context: Context) => {
    try {
        // Get secret with caching (default TTL: 5 seconds)
        const secretValue = await getSecret('my-secret-name');

        // Get secret with custom TTL
        const secretWithTtl = await getSecret('my-secret-name', { maxAge: 300 });

        // Get secret and transform JSON
        const secretJson = await getSecret('my-json-secret', { transform: 'json' });

        logger.info('Successfully retrieved secrets');

        return {
            statusCode: 200,
            body: 'Successfully retrieved secrets'
        };

    } catch (error) {
        logger.error('Error retrieving secret', { error });
        return {
            statusCode: 500,
            body: `Error: ${error}`
        };
    }
};
```

Java

> **Note**
>
> Untuk contoh lengkap dan petunjuk penyiapan, lihat dokumentasi utilitas Parameter.

Mengambil rahasia dari Secrets Manager dengan Powertools untuk utilitas AWS Lambda Parameter.

```
import software.amazon.lambda.powertools.logging.Logging;
import software.amazon.lambda.powertools.parameters.SecretsProvider;
import software.amazon.lambda.powertools.parameters.ParamManager;
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;

public class SecretHandler implements RequestHandler<Object, String> {

    private final SecretsProvider secretsProvider =
 ParamManager.getSecretsProvider();

    @Logging
    @Override
    public String handleRequest(Object input, Context context) {
        try {
            // Get secret with caching (default TTL: 5 seconds)
            String secretValue = secretsProvider.get("my-secret-name");

            // Get secret with custom TTL (300 seconds)
            String secretWithTtl = secretsProvider.withMaxAge(300).get("my-secret-
name");

            // Get secret and transform JSON
            MySecret secretJson = secretsProvider.get("my-json-secret",
 MySecret.class);

            return "Successfully retrieved secrets";

        } catch (Exception e) {
            return "Error retrieving secret: " + e.getMessage();
        }
    }
```

```
    public static class MySecret {
        // Define your secret structure here
    }
}
```

.NET

> **Note**
>
> Untuk contoh lengkap dan petunjuk penyiapan, lihat dokumentasi utilitas Parameter.

Mengambil rahasia dari Secrets Manager dengan Powertools untuk utilitas AWS Lambda Parameter.

```
using AWS.Lambda.Powertools.Logging;
using AWS.Lambda.Powertools.Parameters;
using Amazon.Lambda.Core;

[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeriali

public class Function
{
    private readonly ISecretsProvider _secretsProvider;

    public Function()
    {
        _secretsProvider = ParametersManager.SecretsProvider;
    }

    [Logging]
    public async Task<string> FunctionHandler(object input, ILambdaContext context)
    {
        try
        {
            // Get secret with caching (default TTL: 5 seconds)
            var secretValue = await _secretsProvider.GetAsync("my-secret-name");

            // Get secret with custom TTL
            var secretWithTtl = await
 _secretsProvider.WithMaxAge(TimeSpan.FromMinutes(5))
                .GetAsync("my-secret-name");
```

```
            // Get secret and transform JSON
            var secretJson = await _secretsProvider.GetAsync<MySecret>("my-json-
secret");

            return "Successfully retrieved secrets";
        }
        catch (Exception e)
        {
            return $"Error retrieving secret: {e.Message}";
        }
    }

    public class MySecret
    {
        // Define your secret structure here
    }
}
```

Pengaturan dan izin

Untuk menggunakan utilitas Parameter, Anda perlu:

1. Instal Powertools AWS Lambda untuk runtime Anda. Lihat perinciannya di Powertools untuk AWS Lambda.
2. Tambahkan izin IAM yang diperlukan ke peran eksekusi fungsi Anda. Lihat Mengelola izin di AWS Lambda untuk detailnya.
3. Konfigurasikan pengaturan opsional apa pun melalui variabel lingkungan.

Izin IAM yang diperlukan sama dengan pendekatan ekstensi. Utilitas secara otomatis menangani caching dan panggilan API ke Secrets Manager berdasarkan konfigurasi Anda.

Menggunakan Lambda dengan Amazon SQS

> **Note**
>
> Jika Anda ingin mengirim data ke target selain Fungsi Peristiwa Lambda atau memperkaya data sebelum mengirimnya, lihat Amazon Pipes. EventBridge

Anda dapat menggunakan Fungsi Peristiwa Lambda untuk memproses pesan dalam antrean Amazon Simple Queue Service (Amazon SQS). Lambda mendukung antrian standar dan antrianfirst-in, first-out (FIFO) untuk pemetaan sumber acara. Anda juga dapat menggunakan mode yang disediakan untuk mengalokasikan sumber daya polling khusus untuk pemetaan sumber peristiwa Amazon SQS Anda. Fungsi Acara Lambda dan antrean Amazon SQS harus Wilayah AWS sama, meskipun bisa berbeda. Akun AWS

Untuk fungsi contoh yang memproses pesan Amazon SQS, lihat. the section called “Tutorial” Untuk pemrosesan lintas akun, lihatthe section called “Tutorial lintas akun SQS”. Untuk daftar lengkap opsi konfigurasi, lihatthe section called “Parameter”.

Saat memproses pesan Amazon SQS, Anda perlu menerapkan logika respons batch sebagian untuk mencegah pesan yang berhasil diproses dicoba ulang ketika beberapa pesan dalam batch gagal. Utilitas Prosesor Batch dari Powertools untuk AWS Lambda menyederhanakan implementasi ini dengan secara otomatis menangani logika respons sebagian batch, mengurangi waktu pengembangan dan meningkatkan keandalan.

Topik

- Memahami perilaku polling dan batching untuk pemetaan sumber peristiwa Amazon SQS
- Menggunakan mode yang disediakan dengan pemetaan sumber peristiwa Amazon SQS
- Mengonfigurasi mode yang disediakan untuk pemetaan sumber peristiwa Amazon SQS
- Contoh acara pesan antrian standar
- Contoh acara pesan antrian FIFO
- Membuat dan mengonfigurasi pemetaan sumber peristiwa Amazon SQS
- Mengkonfigurasi perilaku penskalaan untuk pemetaan sumber peristiwa SQS
- Menangani kesalahan untuk sumber peristiwa SQS di Lambda
- Parameter Lambda untuk pemetaan sumber peristiwa Amazon SQS

- Menggunakan pemfilteran peristiwa dengan sumber peristiwa Amazon SQS
- Tutorial: Menggunakan Lambda dengan Amazon SQS
- Tutorial: Menggunakan antrian Amazon SQS lintas akun sebagai sumber acara

Memahami perilaku polling dan batching untuk pemetaan sumber peristiwa Amazon SQS

Dengan pemetaan sumber peristiwa Amazon SQS, Lambda melakukan polling antrian dan memanggil fungsi Anda secara sinkron dengan suatu peristiwa. Setiap acara dapat berisi sekumpulan beberapa pesan dari antrian. Lambda menerima peristiwa ini satu batch pada satu waktu, dan memanggil fungsi Anda sekali untuk setiap batch. Ketika fungsi Anda berhasil memproses suatu batch, Lambda menghapus pesannya dari antrean.

Saat Lambda menerima batch, pesan tetap berada dalam antrean tetapi disembunyikan selama batas waktu visibilitas antrian. Jika fungsi Anda berhasil memproses semua pesan dalam batch, Lambda menghapus pesan dari antrian. Secara default, jika fungsi Anda mengalami kesalahan saat memproses batch, semua pesan dalam kumpulan tersebut akan terlihat lagi dalam antrian setelah batas waktu visibilitas berakhir. Untuk alasan ini, kode fungsi Anda harus dapat memproses pesan yang sama beberapa kali tanpa efek samping yang tidak diinginkan.

> ⚠ Warning
>
> Pemetaan sumber peristiwa Lambda memproses setiap peristiwa setidaknya sekali, dan pemrosesan duplikat catatan dapat terjadi. Untuk menghindari potensi masalah yang terkait dengan duplikat peristiwa, kami sangat menyarankan agar Anda membuat kode fungsi Anda idempoten. Untuk mempelajari lebih lanjut, lihat Bagaimana cara membuat fungsi Lambda saya idempoten di Pusat Pengetahuan. AWS

Untuk mencegah Lambda memproses pesan beberapa kali, Anda dapat mengonfigurasi pemetaan sumber peristiwa untuk menyertakan kegagalan item batch dalam respons fungsi, atau Anda dapat menggunakan DeleteMessageAPI untuk menghapus pesan dari antrian saat Fungsi Peristiwa Lambda berhasil memprosesnya.

Untuk informasi selengkapnya tentang parameter konfigurasi yang didukung Lambda untuk pemetaan sumber peristiwa SQS, lihat. the section called “Membuat pemetaan sumber acara SQS”

Menggunakan mode yang disediakan dengan pemetaan sumber peristiwa Amazon SQS

Untuk beban kerja di mana Anda perlu menyempurnakan throughput pemetaan sumber peristiwa, Anda dapat menggunakan mode yang disediakan. Dalam mode yang disediakan, Anda menentukan batas minimum dan maksimum untuk jumlah poller acara yang disediakan. Poller acara yang disediakan ini didedikasikan untuk pemetaan sumber acara Anda, dan dapat menangani lonjakan pesan yang tidak terduga melalui penskalaan otomatis responsif. Pemetaan sumber peristiwa Amazon SQS yang dikonfigurasi dengan Mode Tertentu menskalakan 3x lebih cepat (hingga 1.000 pemanggilan bersamaan per menit) dan mendukung konkurensi 80x lebih tinggi (hingga 100.000 pemanggilan bersamaan) daripada kemampuan pemetaan sumber peristiwa Amazon SQS default.

Kami menyarankan Anda menggunakan mode yang disediakan untuk beban kerja berbasis peristiwa Amazon SQS yang memiliki persyaratan kinerja yang ketat, seperti perusahaan jasa keuangan yang memproses umpan data pasar, platform e-commerce yang memberikan rekomendasi yang dipersonalisasi secara real-time, dan perusahaan game yang mengelola interaksi pemain langsung. Menggunakan mode yang disediakan menimbulkan biaya tambahan. Untuk harga terperinci, lihat AWS Lambda harga.

Setiap poller peristiwa dalam mode yang disediakan dapat menangani hingga 1 throughput, hingga 10 MB/s pemanggilan bersamaan, atau hingga 10 panggilan API polling Amazon SQS per detik. Rentang nilai yang diterima untuk jumlah minimum poller peristiwa (MinimumPollers) adalah antara 2 dan 200, dengan default 2. Rentang nilai yang diterima untuk jumlah maksimum poller acara (MaximumPollers) adalah antara 2 dan 10.000, dengan default 200. MaximumPollers harus lebih besar dari atau sama dengan MinimumPollers.

Menentukan poller acara yang diperlukan

Untuk memperkirakan jumlah poller peristiwa yang diperlukan untuk memastikan kinerja pemrosesan pesan yang optimal saat menggunakan mode yang disediakan untuk SQS ESM, kumpulkan metrik berikut untuk aplikasi Anda: peristiwa SQS puncak per detik yang memerlukan pemrosesan latensi rendah, ukuran payload peristiwa SQS rata-rata, durasi Fungsi Peristiwa Lambda rata-rata, dan ukuran batch yang dikonfigurasi.

Pertama, Anda dapat memperkirakan jumlah peristiwa SQS per detik (EPS) yang didukung oleh poller acara untuk beban kerja Anda menggunakan rumus berikut:

```
EPS per event poller =
```

```
        minimum(
            ceiling(1024 / average event size in KB),
            ceiling(10 / average function duration in seconds) * batch size,
            min(100, 10 * batch size)
                )
```

Kemudian, Anda dapat menghitung jumlah poller minimum yang diperlukan menggunakan rumus di bawah ini. Perhitungan ini memastikan Anda menyediakan kapasitas yang cukup untuk menangani persyaratan lalu lintas puncak Anda.

```
Required event pollers = (Peak number of events per second in Queue) / EPS per event
 poller
```

Pertimbangkan beban kerja dengan ukuran batch default 10, ukuran acara rata-rata 3 KB, durasi fungsi rata-rata 100 ms, dan persyaratan untuk menangani 1.000 peristiwa per detik. Dalam skenario ini, setiap poller acara mendukung sekitar 100 peristiwa per detik (EPS). Oleh karena itu, kami menyarankan Anda menetapkan poller minimum ke 10 untuk menangani persyaratan lalu lintas puncak Anda secara memadai. Jika beban kerja Anda memiliki karakteristik yang sama tetapi dengan durasi fungsi rata-rata 1 detik, setiap poller hanya mendukung 10 EPS, mengharuskan Anda mengonfigurasi 100 poller minimum untuk mendukung 1.000 peristiwa per detik pada latensi rendah.

Sebaiknya gunakan ukuran batch default 10 atau lebih tinggi untuk memaksimalkan efisiensi poller acara mode yang disediakan. Ukuran batch yang lebih tinggi memungkinkan setiap poller memproses lebih banyak acara per pemanggilan, untuk meningkatkan throughput dan efisiensi biaya. Saat merencanakan kapasitas poller acara Anda, perhitungkan potensi lonjakan lalu lintas dan pertimbangkan untuk menetapkan nilai MinimumPollers Anda sedikit lebih tinggi dari minimum yang dihitung untuk menyediakan buffer. Selain itu, pantau karakteristik beban kerja Anda dari waktu ke waktu, karena perubahan ukuran pesan, durasi fungsi, atau pola lalu lintas mungkin memerlukan penyesuaian konfigurasi poller acara Anda untuk mempertahankan kinerja dan efisiensi biaya yang optimal. Untuk perencanaan kapasitas yang tepat, kami sarankan untuk menguji beban kerja spesifik Anda untuk menentukan EPS aktual yang dapat didorong oleh setiap poller peristiwa.

Mengonfigurasi mode yang disediakan untuk pemetaan sumber peristiwa Amazon SQS

Anda dapat mengonfigurasi mode yang disediakan untuk pemetaan sumber peristiwa Amazon SQS menggunakan konsol atau API Lambda.

Untuk mengonfigurasi mode yang disediakan untuk pemetaan sumber peristiwa Amazon SQS yang ada (konsol)

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih fungsi dengan pemetaan sumber peristiwa Amazon SQS yang ingin Anda konfigurasikan untuk mode yang disediakan.
3. Pilih Konfigurasi, lalu pilih Pemicu.
4. Pilih pemetaan sumber peristiwa Amazon SQS yang ingin Anda konfigurasikan untuk mode yang disediakan, lalu pilih Edit.
5. Di bawah Konfigurasi pemetaan sumber peristiwa, pilih Konfigurasi mode yang disediakan.
   - Untuk poller acara Minimum, masukkan nilai antara 2 dan 200. Jika Anda tidak menentukan nilai, Lambda memilih nilai default 2.
   - Untuk poller acara Maksimum, masukkan nilai antara 2 dan 10.000. Nilai ini harus lebih besar dari atau sama dengan nilai Anda untuk poller acara Minimum. Jika Anda tidak menentukan nilai, Lambda memilih nilai default 200.
6. Pilih Simpan.

Anda dapat mengonfigurasi mode yang disediakan secara terprogram menggunakan objek di. `ProvisionedPollerConfig` `EventSourceMappingConfiguration` Misalnya, perintah `UpdateEventSourceMapping` CLI berikut mengkonfigurasi `MinimumPollers` nilai 5, dan `MaximumPollers` nilai 100.

```
aws lambda update-event-source-mapping \
    --uuid a1b2c3d4-5678-90ab-cdef-EXAMPLE11111 \
    --provisioned-poller-config '{"MinimumPollers": 5, "MaximumPollers": 100}'
```

Setelah mengonfigurasi mode yang disediakan, Anda dapat mengamati penggunaan poller peristiwa untuk beban kerja Anda dengan memantau metrik. `ProvisionedPollers` Untuk informasi selengkapnya, lihat Metrik pemetaan sumber peristiwa.

Untuk menonaktifkan mode yang disediakan dan kembali ke mode default (sesuai permintaan), Anda dapat menggunakan perintah CLI `UpdateEventSourceMapping` berikut:

```
aws lambda update-event-source-mapping \
    --uuid a1b2c3d4-5678-90ab-cdef-EXAMPLE11111 \
```

```
    --provisioned-poller-config '{}'
```

> **Note**
>
> Mode yang disediakan tidak dapat digunakan bersamaan dengan pengaturan konkurensi maksimum. Saat menggunakan mode yang disediakan, Anda mengontrol konkurensi maksimum melalui jumlah maksimum poller acara.

Untuk informasi selengkapnya tentang mengonfigurasi mode yang disediakan, lihat. the section called “Buat pemetaan”

Contoh acara pesan antrian standar

Example Kejadian pesan Amazon SQS (antrean standar)


```
{
    "Records": [
        {
            "messageId": "059f36b4-87a3-44ab-83d2-661975830a7d",
            "receiptHandle": "AQEBwJnKyrHigUMZj6rYigCgxlaS3SLy0a...",
            "body": "Test message.",
            "attributes": {
                "ApproximateReceiveCount": "1",
                "SentTimestamp": "1545082649183",
                "SenderId": "AIDAIENQZJOLO23YVJ4VO",
                "ApproximateFirstReceiveTimestamp": "1545082649185"
            },
            "messageAttributes": {
                "myAttribute": {
                    "stringValue": "myValue",
                    "stringListValues": [],
                    "binaryListValues": [],
                    "dataType": "String"
                }
            },
            "md5OfBody": "e4e68fb7bd0e697a0ae8f1bb342846b3",
            "eventSource": "aws:sqs",
            "eventSourceARN": "arn:aws:sqs:us-east-2:123456789012:my-queue",
            "awsRegion": "us-east-2"
        },
        {
```

```
            "messageId": "2e1424d4-f796-459a-8184-9c92662be6da",
            "receiptHandle": "AQEBzWwaftRI0KuVm4tP+/7q1rGgNqicHq...",
            "body": "Test message.",
            "attributes": {
                "ApproximateReceiveCount": "1",
                "SentTimestamp": "1545082650636",
                "SenderId": "AIDAIENQZJOLO23YVJ4VO",
                "ApproximateFirstReceiveTimestamp": "1545082650649"
            },
            "messageAttributes": {},
            "md5OfBody": "e4e68fb7bd0e697a0ae8f1bb342846b3",
            "eventSource": "aws:sqs",
            "eventSourceARN": "arn:aws:sqs:us-east-2:123456789012:my-queue",
            "awsRegion": "us-east-2"
        }
    ]
}
```

Secara default, Lambda melakukan polling hingga 10 pesan dalam antrian sekaligus dan mengirimkan batch tersebut ke fungsi Anda. Untuk menghindari menjalankan fungsi dengan sejumlah kecil catatan, Anda dapat mengonfigurasi sumber peristiwa untuk menyangga catatan hingga 5 menit dengan mengonfigurasi jendela batch. Sebelum menjalankan fungsi, Lambda melanjutkan polling pesan dari antrian standar hingga jendela batch kedaluwarsa, kuota ukuran payload pemanggilan tercapai, atau ukuran batch maksimum yang dikonfigurasi tercapai.

Jika Anda menggunakan jendela batch dan antrian SQS Anda berisi lalu lintas yang sangat rendah, Lambda mungkin menunggu hingga 20 detik sebelum menjalankan fungsi Anda. Ini benar bahkan jika Anda mengatur jendela batch lebih rendah dari 20 detik.

> **Note**
>
> Di Java, Anda mungkin mengalami kesalahan pointer null saat deserialisasi JSON. Ini bisa disebabkan oleh bagaimana kasus “Records” dan “EventSourcEarn” dikonversi oleh pemeta objek JSON.

Contoh acara pesan antrian FIFO

Untuk antrean FIFO, rekaman berisi atribut tambahan yang terkait dengan deduplikasi dan pengurutan.

Example Kejadian pesan Amazon SQS (antrean FIFO)

```
{
    "Records": [
        {
            "messageId": "11d6ee51-4cc7-4302-9e22-7cd8afdaadf5",
            "receiptHandle": "AQEBBX8nesZEXmkhsmZeyIE8iQAMig7qw...",
            "body": "Test message.",
            "attributes": {
                "ApproximateReceiveCount": "1",
                "SentTimestamp": "1573251510774",
                "SequenceNumber": "18849496460467696128",
                "MessageGroupId": "1",
                "SenderId": "AIDAIO23YVJENQZJOL4VO",
                "MessageDeduplicationId": "1",
                "ApproximateFirstReceiveTimestamp": "1573251510774"
            },
            "messageAttributes": {},
            "md5OfBody": "e4e68fb7bd0e697a0ae8f1bb342846b3",
            "eventSource": "aws:sqs",
            "eventSourceARN": "arn:aws:sqs:us-east-2:123456789012:fifo.fifo",
            "awsRegion": "us-east-2"
        }
    ]
}
```

Membuat dan mengonfigurasi pemetaan sumber peristiwa Amazon SQS

Untuk memproses pesan Amazon SQS dengan Lambda, konfigurasikan antrian Anda dengan pengaturan yang sesuai, lalu buat pemetaan sumber peristiwa Lambda.

Topik

- Mengonfigurasi antrean untuk digunakan dengan Lambda
- Menyiapkan izin peran eksekusi Lambda
- Membuat pemetaan sumber acara SQS

Mengonfigurasi antrean untuk digunakan dengan Lambda

Jika Anda belum memiliki antrean Amazon SQS yang ada, buat antrean untuk berfungsi sebagai sumber acara untuk Fungsi Peristiwa Lambda Anda. Fungsi Acara Lambda dan antrean Amazon SQS harus Wilayah AWS sama, meskipun bisa berbeda. Akun AWS

Untuk memungkinkan waktu fungsi Anda memproses setiap kumpulan rekaman, atur batas waktu visibilitas antrian sumber menjadi setidaknya enam kali batas waktu konfigurasi pada fungsi Anda. Waktu tambahan memungkinkan Lambda untuk mencoba lagi jika fungsi Anda dibatasi saat memproses batch sebelumnya.

> **Note**
>
> Batas waktu fungsi Anda harus kurang dari atau sama dengan batas waktu visibilitas antrian. Lambda memvalidasi persyaratan ini saat Anda membuat atau memperbarui pemetaan sumber peristiwa dan mengembalikan kesalahan jika batas waktu fungsi melebihi batas waktu visibilitas antrian.

Secara default, jika Lambda menemukan kesalahan di titik mana pun saat memproses batch, semua pesan dalam batch tersebut kembali ke antrian. Setelah batas waktu visibilitas, pesan menjadi terlihat oleh Lambda lagi. Anda dapat mengonfigurasi pemetaan sumber peristiwa untuk menggunakan respons batch sebagian agar hanya mengembalikan pesan yang gagal kembali ke antrian.

Selain itu, jika fungsi Anda gagal memproses pesan beberapa kali, Amazon SQS dapat mengirimkannya ke antrian huruf mati. Sebaiknya setel kebijakan redrive antrian sumber Anda ke setidaknya 5. `maxReceiveCount` Ini memberi Lambda beberapa kesempatan untuk mencoba lagi sebelum mengirim pesan yang gagal langsung ke antrian surat mati.

Menyiapkan izin peran eksekusi Lambda

Kebijakan AWSLambdaSQSQueueExecutionRole AWS terkelola mencakup izin yang perlu dibaca Lambda dari antrean Amazon SQS Anda. Anda dapat menambahkan kebijakan terkelola ini ke peran eksekusi fungsi Anda.

Secara opsional, jika Anda menggunakan antrean terenkripsi, Anda juga perlu menambahkan izin berikut ke peran eksekusi Anda:

- kms:Decrypt

Membuat pemetaan sumber acara SQS

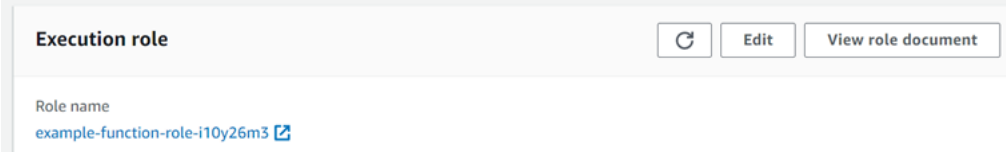
Buat pemetaan sumber acara untuk memberi tahu Lambda agar mengirim item dari antrian Anda ke Fungsi Peristiwa Lambda. Anda dapat membuat beberapa pemetaan sumber kejadian untuk

memproses item dari beberapa antrean dengan satu fungsi. Ketika Lambda memanggil fungsi target, kejadian dapat berisi beberapa item, hingga ukuran batch maksimum yang dapat dikonfigurasi.

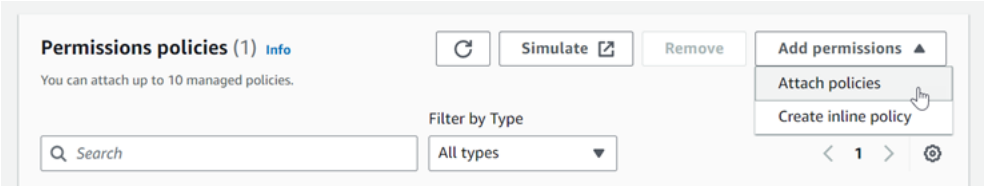
Untuk mengonfigurasi fungsi agar dibaca dari Amazon SQS, lampirkan kebijakan AWSLambdaSQSQueueExecutionRole AWS terkelola ke peran eksekusi Anda. Kemudian, buat pemetaan sumber peristiwa SQS dari konsol menggunakan langkah-langkah berikut.

Untuk menambahkan izin dan membuat pemicu

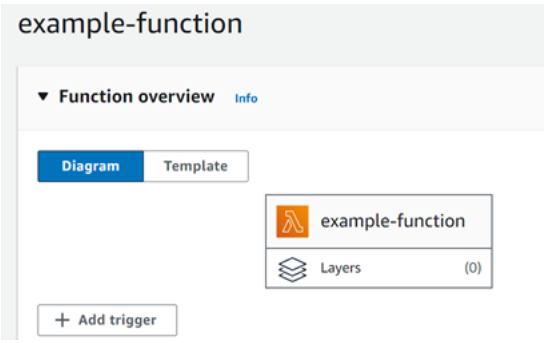
1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih nama sebuah fungsi.
3. Pilih tab Konfigurasi, lalu pilih Izin.
4. Di bawah Nama peran, pilih tautan ke peran eksekusi Anda. Tautan ini membuka peran di konsol IAM.



5. Pilih Tambahkan izin, lalu pilih Lampirkan kebijakan.



6. Di bidang pencarian, masukkan`AWSLambdaSQSQueueExecutionRole`. Tambahkan kebijakan ini ke peran eksekusi Anda. Ini adalah kebijakan AWS terkelola yang berisi izin yang perlu dibaca fungsi Anda dari antrean Amazon SQS. Untuk informasi selengkapnya tentang kebijakan ini, lihat AWSLambdaSQSQueueExecutionRoledi Referensi Kebijakan AWS Terkelola.
7. Kembali ke fungsi Anda di konsol Lambda. Di bagian Gambaran umum fungsi, pilih Tambah pemicu.



8. Pilih jenis pemicu.
9. Konfigurasikan opsi yang diperlukan, lalu pilih Tambah.

Lambda mendukung opsi konfigurasi berikut untuk sumber peristiwa Amazon SQS:

Antrian SQS

Antrian Amazon SQS untuk membaca catatan dari. Fungsi Acara Lambda dan antrean Amazon SQS harus Wilayah AWS sama, meskipun bisa berbeda. Akun AWS

Aktifkan pemicu

Status pemetaan sumber acara. Aktifkan pemicu dipilih secara default.

Ukuran batch

Jumlah maksimum catatan untuk dikirim ke fungsi di setiap batch. Untuk antrian standar, ini bisa mencapai 10.000 catatan. Untuk antrian FIFO, maksimum adalah 10. Untuk ukuran batch lebih dari 10, Anda juga harus mengatur batch window (`MaximumBatchingWindowInSeconds`) setidaknya 1 detik.

Konfigurasikan batas waktu fungsi Anda untuk memberikan waktu yang cukup untuk memproses seluruh kumpulan item. Jika item membutuhkan waktu lama untuk diproses, pilih ukuran batch yang lebih kecil. Ukuran batch yang besar dapat meningkatkan efisiensi beban kerja yang sangat cepat atau memiliki banyak overhead. Jika Anda mengonfigurasi konkurensi cadangan pada fungsi Anda, setel minimal lima eksekusi bersamaan untuk mengurangi kemungkinan kesalahan pelambatan saat Lambda memanggil fungsi Anda.

Lambda meneruskan semua catatan dalam batch ke fungsi dalam satu panggilan, selama ukuran total peristiwa tidak melebihi kuota ukuran payload pemanggilan untuk pemanggilan sinkron (6 MB). Baik Lambda dan Amazon SQS menghasilkan metadata untuk setiap catatan. Metadata tambahan ini dihitung terhadap ukuran muatan total dan dapat menyebabkan jumlah total catatan yang dikirim dalam batch menjadi lebih rendah dari ukuran batch yang dikonfigurasi. Bidang metadata yang dikirimkan Amazon SQS dapat bervariasi panjangnya. Untuk informasi selengkapnya tentang bidang metadata Amazon SQS, lihat dokumentasi operasi ReceiveMessageAPI di Referensi API Layanan Antrian Sederhana Amazon.

Jendela batch

Jumlah waktu maksimum untuk mengumpulkan rekaman sebelum memanggil fungsi, dalam hitungan detik. Ini hanya berlaku untuk antrian standar.

Jika Anda menggunakan jendela batch lebih dari 0 detik, Anda harus memperhitungkan peningkatan waktu pemrosesan dalam batas waktu visibilitas antrian Anda. Sebaiknya atur batas waktu visibilitas antrian Anda menjadi enam kali batas waktu Fungsi Acara Anda, ditambah nilai. `MaximumBatchingWindowInSeconds` Ini memungkinkan waktu bagi Fungsi Peristiwa Lambda Anda untuk memproses setiap kumpulan peristiwa dan mencoba lagi jika terjadi kesalahan pelambatan.

Ketika pesan tersedia, Lambda mulai memproses pesan dalam batch. Lambda mulai memproses lima batch sekaligus dengan lima pemanggilan fungsi Anda secara bersamaan. Jika pesan masih tersedia, Lambda menambahkan hingga 300 pemanggilan fungsi secara bersamaan dalam satu menit, hingga maksimum 1.250 pemanggilan bersamaan.

Saat menggunakan mode yang disediakan, setiap poller peristiwa dapat menangani hingga 1 throughput, hingga 10 MB/s pemanggilan bersamaan, atau hingga 10 panggilan API polling Amazon SQS per detik. Lambda menskalakan jumlah poller peristiwa antara minimum dan maksimum yang dikonfigurasi, dengan cepat menambahkan hingga 1.000 pemanggilan bersamaan per menit untuk menyediakan pemrosesan latensi rendah pada peristiwa Amazon SQS Anda. Anda mengontrol penskalaan dan konkurensi melalui pengaturan poller peristiwa minimum dan maksimum ini. Untuk mempelajari lebih lanjut tentang penskalaan dan konkurensi Fungsi Acara, lihat. Memahami penskalaan Fungsi Acara Lambda

Untuk memproses lebih banyak pesan, Anda dapat mengoptimalkan Fungsi Peristiwa Lambda untuk throughput yang lebih tinggi. Untuk informasi selengkapnya, lihat Memahami cara AWS Lambda menskalakan dengan antrian standar Amazon SQS.

Kriteria filter

Tambahkan kriteria filter untuk mengontrol peristiwa yang dikirim Lambda ke fungsi Anda untuk diproses. Untuk informasi selengkapnya, lihat Kontrol peristiwa mana yang dikirim Lambda ke Fungsi Acara Anda.

Konkurensi maksimum

Jumlah maksimum fungsi bersamaan yang dapat dipanggil sumber peristiwa. Tidak dapat digunakan dengan Mode Penyediaan diaktifkan. Untuk informasi selengkapnya, lihat Mengkonfigurasi konkurensi maksimum untuk sumber peristiwa Amazon SQS.

Mode yang Disediakan

Saat diaktifkan, alokasikan sumber daya polling khusus untuk pemetaan sumber acara Anda. Anda dapat mengonfigurasi jumlah poller acara minimum (2-200) dan maksimum (2-2000). Setiap

event poller dapat menangani hingga 1 MB/sec throughput, hingga 10 pemanggilan bersamaan, atau hingga 10 panggilan API polling Amazon SQS per detik.

> **Note**
>
> Catatan: Anda tidak dapat menggunakan Provisioned Mode dan Maximum concurrency bersama-sama. Saat Mode Penyediaan diaktifkan, gunakan pengaturan poller maksimum untuk mengontrol konkurensi.

Mengkonfigurasi perilaku penskalaan untuk pemetaan sumber peristiwa SQS

Anda dapat mengontrol perilaku penskalaan pemetaan sumber peristiwa Amazon SQS baik melalui pengaturan konkurensi maksimum atau dengan mengaktifkan mode yang disediakan. Ini adalah opsi yang saling eksklusif.

Secara default, Lambda secara otomatis menskalakan poller peristiwa berdasarkan volume pesan. Saat mengaktifkan mode yang disediakan, Anda mengalokasikan jumlah minimum dan maksimum sumber daya polling khusus yang tetap siap menangani pola lalu lintas yang diharapkan. Anda dapat mengoptimalkan kinerja pemetaan sumber acara dengan dua cara:

- Mode standar (Default): Lambda secara otomatis mengelola penskalaan, dimulai dengan sejumlah kecil poller dan penskalaan naik atau turun berdasarkan beban kerja.
- Mode yang disediakan: Anda mengonfigurasi sumber daya polling khusus dengan batas minimum dan maksimum, memungkinkan penskalaan 3 kali lebih cepat dan kapasitas pemrosesan hingga 16 kali lebih tinggi.

Untuk antrian standar, Lambda menggunakan polling panjang untuk melakukan polling antrian hingga menjadi aktif. Ketika pesan tersedia, Lambda mulai memproses lima batch sekaligus dengan lima pemanggilan bersamaan dari Fungsi Acara Anda. Jika pesan masih tersedia, Lambda meningkatkan jumlah proses yang membaca batch hingga 300 pemanggilan bersamaan per menit. Jumlah maksimum pemanggilan yang dapat diproses oleh pemetaan sumber peristiwa secara bersamaan adalah 1.250. Ketika lalu lintas rendah, Lambda menskalakan kembali pemrosesan ke lima pemanggilan bersamaan, dan dapat mengoptimalkan sedikitnya 2 pemanggilan bersamaan untuk mengurangi panggilan Amazon SQS dan biaya terkait. Namun, pengoptimalan ini tidak tersedia saat Anda mengaktifkan pengaturan konkurensi maksimum.

Untuk antrean FIFO, Lambda mengirimkan pesan ke fungsi Anda sesuai urutan penerimaan. Saat Anda mengirimkan pesan ke antrean FIFO, Anda menentukan ID grup pesan. Amazon SQS memastikan bahwa pesan dalam grup yang sama dikirim ke Lambda secara berurutan. Saat Lambda membaca pesan Anda ke dalam batch, setiap batch mungkin berisi pesan dari lebih dari satu grup pesan, tetapi urutan pesan tetap dipertahankan. Jika fungsi Anda menampilkan kesalahan, fungsi akan mencoba semua percobaan ulang pada pesan yang terpengaruh sebelum Lambda menerima pesan tambahan dari grup yang sama.

Saat menggunakan mode yang disediakan, setiap poller peristiwa dapat menangani hingga 1 throughput, hingga 10 MB/sec pemanggilan bersamaan, atau hingga 10 panggilan API polling Amazon SQS per detik. Lambda menskalakan jumlah poller peristiwa antara minimum dan maksimum yang dikonfigurasi, dengan cepat menambahkan hingga 1.000 konkurensi per menit untuk menyediakan pemrosesan peristiwa Amazon SQS Anda yang konsisten dan latensi rendah. Menggunakan mode yang disediakan menimbulkan biaya tambahan. Untuk harga terperinci, lihat AWS Lambda harga.

Setiap poller acara menggunakan polling panjang ke antrean SQS Anda hingga 10 polling per detik, yang menimbulkan biaya permintaan SQS API. Lihat harga Amazon SQS untuk detailnya. Anda mengontrol penskalaan dan konkurensi melalui pengaturan poller peristiwa minimum dan maksimum ini, daripada menggunakan pengaturan konkurensi maksimum, karena opsi ini tidak dapat digunakan bersama.

> **Note**
>
> Anda tidak dapat menggunakan pengaturan konkurensi maksimum dan mode yang disediakan secara bersamaan. Saat mode yang disediakan diaktifkan, Anda mengontrol penskalaan dan konkurensi pemetaan sumber peristiwa Amazon SQS melalui jumlah minimum dan maksimum poller peristiwa.

Mengkonfigurasi konkurensi maksimum untuk sumber peristiwa Amazon SQS

Anda dapat menggunakan setelan konkurensi maksimum untuk mengontrol perilaku penskalaan untuk sumber peristiwa SQS Anda. Perhatikan bahwa konkurensi maksimum tidak dapat digunakan dengan mode yang disediakan diaktifkan. Setelan konkurensi maksimum membatasi jumlah instance bersamaan dari fungsi yang dapat dipanggil oleh sumber peristiwa Amazon SQS. Konkurensi maksimum adalah pengaturan tingkat sumber acara. Jika Anda memiliki beberapa sumber peristiwa

Amazon SQS yang dipetakan ke satu fungsi, setiap sumber peristiwa dapat memiliki setelan konkurensi maksimum yang terpisah.

Anda dapat menggunakan konkurensi maksimum untuk mencegah satu antrian menggunakan semua konkurensi cadangan fungsi atau sisa kuota konkurensi akun. Tidak ada biaya untuk mengonfigurasi konkurensi maksimum pada sumber acara Amazon SQS.

Yang penting, konkurensi maksimum dan konkurensi cadangan adalah dua pengaturan independen. Jangan setel konkurensi maksimum yang lebih tinggi dari konkurensi cadangan fungsi. Jika Anda mengonfigurasi konkurensi maksimum, pastikan konkurensi cadangan fungsi Anda lebih besar dari atau sama dengan total konkurensi maksimum untuk semua sumber peristiwa Amazon SQS pada fungsi tersebut. Jika tidak, Lambda mungkin membatasi pesan Anda.

Jika kuota konkurensi akun Anda disetel ke nilai default 1.000, pemetaan sumber peristiwa Amazon SQS dapat menskalakan untuk memanggil instance fungsi hingga nilai ini, kecuali Anda menentukan konkurensi maksimum.

Jika Anda menerima peningkatan kuota konkurensi default akun Anda, Lambda mungkin tidak dapat menjalankan instance fungsi bersamaan hingga kuota baru Anda. Secara default, Lambda dapat menskalakan untuk menjalankan hingga 1.250 instance fungsi bersamaan untuk pemetaan sumber peristiwa Amazon SQS. Jika Anda membutuhkan konkurensi yang lebih tinggi, aktifkan mode yang disediakan. Mode yang disediakan menskalakan hingga 100.000 pemanggilan fungsi bersamaan. Untuk informasi selengkapnya, lihat Menggunakan mode yang disediakan dengan pemetaan sumber peristiwa Amazon SQS.

> **Note**
>
> Untuk antrian FIFO, pemanggilan bersamaan dibatasi oleh jumlah ID grup pesan (`messageGroupId`) atau pengaturan konkurensi maksimum—mana yang lebih rendah. Misalnya, jika Anda memiliki enam ID grup pesan dan konkurensi maksimum disetel ke 10, fungsi Anda dapat memiliki maksimal enam pemanggilan bersamaan.

Anda dapat mengonfigurasi konkurensi maksimum pada pemetaan sumber peristiwa Amazon SQS baru dan yang sudah ada.

Konfigurasikan konkurensi maksimum menggunakan konsol Lambda

1. Buka halaman Fungsi Acara konsol Lambda.

2. Pilih nama sebuah fungsi.
3. Di bawah Ikhtisar fungsi, pilih SQS. Ini membuka tab Konfigurasi.
4. Pilih pemicu Amazon SQS dan pilih Edit.
5. Untuk Konkurensi maksimum, masukkan angka antara 2 dan 1.000. Untuk mematikan konkurensi maksimum, biarkan kotak kosong.
6. Pilih Simpan.

Konfigurasikan konkurensi maksimum menggunakan AWS Command Line Interface (AWS CLI)

Gunakan perintah update-event-source-mapping dengan opsi. `--scaling-config` Contoh:

```
aws lambda update-event-source-mapping \
    --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
    --scaling-config '{"MaximumConcurrency":5}'
```

Untuk mematikan konkurensi maksimum, masukkan nilai kosong untuk`--scaling-config`:

```
aws lambda update-event-source-mapping \
    --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
    --scaling-config "{}"
```

Konfigurasikan konkurensi maksimum menggunakan API Lambda

Gunakan CreateEventSourceMappingatau UpdateEventSourceMappingtindakan dengan ScalingConfigobjek.

Menangani kesalahan untuk sumber peristiwa SQS di Lambda

Untuk menangani kesalahan yang terkait dengan sumber peristiwa SQS, Lambda secara otomatis menggunakan strategi coba lagi dengan strategi backoff. Anda juga dapat menyesuaikan perilaku penanganan kesalahan dengan mengonfigurasi pemetaan sumber peristiwa SQS Anda untuk mengembalikan respons batch sebagian.

Strategi backoff untuk pemanggilan yang gagal

Ketika pemanggilan gagal, Lambda mencoba lagi pemanggilan sambil menerapkan strategi backoff. Strategi backoff sedikit berbeda tergantung pada apakah Lambda mengalami kegagalan karena kesalahan dalam kode fungsi Anda, atau karena pelambatan.

- Jika kode Fungsi Peristiwa Anda menyebabkan kesalahan, Lambda berhenti memproses dan mencoba kembali pemanggilan. Sementara itu, Lambda secara bertahap mundur, mengurangi jumlah konkurensi yang dialokasikan ke pemetaan sumber peristiwa Amazon SQS Anda. Setelah batas waktu visibilitas antrian Anda habis, pesan muncul kembali dalam antrian.
- Jika pemanggilan gagal karena pelambatan, Lambda secara bertahap menghentikan percobaan ulang dengan mengurangi jumlah konkurensi yang dialokasikan ke pemetaan sumber peristiwa Amazon SQS Anda. Lambda terus mencoba lagi pesan hingga stempel waktu pesan melebihi batas waktu visibilitas antrian Anda, di mana Lambda menghapus pesan.

Menerapkan tanggapan batch sebagian

Ketika Fungsi Peristiwa Lambda Anda mengalami kesalahan saat memproses batch, semua pesan dalam kumpulan tersebut akan terlihat lagi dalam antrian secara default, termasuk pesan yang berhasil diproses Lambda. Akibatnya, fungsi Anda akhirnya dapat memproses pesan yang sama beberapa kali.

Untuk menghindari pemrosesan ulang pesan yang berhasil diproses dalam kumpulan yang gagal, Anda dapat mengonfigurasi pemetaan sumber peristiwa agar hanya pesan yang gagal terlihat lagi. Ini disebut respon batch paral. Untuk mengaktifkan respons batch sebagian, tentukan `ReportBatchItemFailures` FunctionResponseTypestindakan saat mengonfigurasi pemetaan sumber peristiwa Anda. Dengan pengaturan ini, fungsi Anda dapat mengembalikan sebagian keberhasilan, yang dapat membantu mengurangi jumlah percobaan ulang yang tidak perlu pada catatan.

> **Note**
>
> Utilitas Prosesor Batch dari Powertools untuk AWS Lambda menangani semua logika respons batch paral secara otomatis. Utilitas ini menyederhanakan penerapan pola pemrosesan batch dan mengurangi kode khusus yang diperlukan untuk menangani kegagalan item batch dengan benar. Ini tersedia untuk Python, Java, TypeScript, dan .NET.

Saat `ReportBatchItemFailures` diaktifkan, Lambda tidak mengurangi polling pesan saat pemanggilan fungsi gagal. Jika Anda mengharapkan beberapa pesan gagal—dan Anda tidak ingin kegagalan tersebut memengaruhi laju pemrosesan pesan—gunakan. `ReportBatchItemFailures`

> **Note**
>
> Ingatlah hal berikut saat menggunakan respons batch sebagian:
>
> - Jika fungsi Anda melempar pengecualian, seluruh batch dianggap gagal total.
> - Jika Anda menggunakan fitur ini dengan antrian FIFO, fungsi Anda harus berhenti memproses pesan setelah kegagalan pertama dan mengembalikan semua pesan yang gagal dan belum diproses. `batchItemFailures` Ini membantu menjaga urutan pesan dalam antrian Anda.

Untuk mengaktifkan pelaporan batch sebagian

1. Tinjau Praktik terbaik untuk menerapkan respons batch sebagian.
2. Jalankan perintah berikut `ReportBatchItemFailures` untuk mengaktifkan fungsi Anda. Untuk mengambil UUID pemetaan sumber peristiwa Anda, jalankan perintah list-event-source-mappings. AWS CLI

```
aws lambda update-event-source-mapping \
--uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
--function-response-types "ReportBatchItemFailures"
```

3. Perbarui kode fungsi Anda untuk menangkap semua pengecualian dan mengembalikan pesan yang gagal dalam respons `batchItemFailures` JSON. `batchItemFailures`Respons harus menyertakan daftar ID pesan, sebagai nilai `itemIdentifier` JSON.

   Misalnya, Anda memiliki sekumpulan lima pesan, dengan ID pesan`id1`,,`id2`, `id3``id4`, dan`id5`. Fungsi Anda berhasil memproses`id1`,`id3`, dan`id5`. Untuk membuat pesan `id2` dan `id4` terlihat lagi dalam antrian Anda, fungsi Anda harus mengembalikan respons berikut:

```
{
  "batchItemFailures": [
        {
            "itemIdentifier": "id2"
        },
        {
            "itemIdentifier": "id4"
        }
    ]
```

```
}
```

Berikut adalah beberapa contoh kode fungsi yang mengembalikan daftar ID pesan yang gagal dalam batch:

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch SQS dengan Lambda menggunakan.NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using Amazon.Lambda.Core;
using Amazon.Lambda.SQSEvents;

// Assembly attribute to enable the Lambda function's JSON input to be
 converted into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJson
namespace sqsSample;

public class Function
{
    public async Task<SQSBatchResponse> FunctionHandler(SQSEvent evnt,
 ILambdaContext context)
    {
        List<SQSBatchResponse.BatchItemFailure> batchItemFailures = new
 List<SQSBatchResponse.BatchItemFailure>();
        foreach(var message in evnt.Records)
        {
            try
            {
                //process your message
                await ProcessMessageAsync(message, context);
            }
```

```
            catch (System.Exception)
            {
                //Add failed message identifier to the batchItemFailures list
                batchItemFailures.Add(new
 SQSBatchResponse.BatchItemFailure{ItemIdentifier=message.MessageId});
            }
        }
        return new SQSBatchResponse(batchItemFailures);
    }

    private async Task ProcessMessageAsync(SQSEvent.SQSMessage message,
 ILambdaContext context)
    {
        if (String.IsNullOrEmpty(message.Body))
        {
            throw new Exception("No Body in SQS Message.");
        }
        context.Logger.LogInformation($"Processed message {message.Body}");
        // TODO: Do interesting work based on the new message
        await Task.CompletedTask;
    }
}
```

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch SQS dengan Lambda menggunakan Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
```

```
 "fmt"
 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

func handler(ctx context.Context, sqsEvent events.SQSEvent)
 (map[string]interface{}, error) {
 batchItemFailures := []map[string]interface{}{}

 for _, message := range sqsEvent.Records {
  if len(message.Body) > 0 {
   // Your message processing condition here
   fmt.Printf("Successfully processed message: %s\n", message.Body)
  } else {
   // Message processing failed
   fmt.Printf("Failed to process message %s\n", message.MessageId)
   batchItemFailures = append(batchItemFailures, map[string]interface{}
{"itemIdentifier": message.MessageId})
  }
 }

 sqsBatchResponse := map[string]interface{}{
  "batchItemFailures": batchItemFailures,
 }
 return sqsBatchResponse, nil
}

func main() {
 lambda.Start(handler)
}
```

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch SQS dengan Lambda menggunakan Java.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.SQSEvent;
import com.amazonaws.services.lambda.runtime.events.SQSBatchResponse;

import java.util.ArrayList;
import java.util.List;

public class ProcessSQSMessageBatch implements RequestHandler<SQSEvent,
 SQSBatchResponse> {
    @Override
    public SQSBatchResponse handleRequest(SQSEvent sqsEvent, Context context)
 {
        List<SQSBatchResponse.BatchItemFailure> batchItemFailures = new
 ArrayList<SQSBatchResponse.BatchItemFailure>();

        for (SQSEvent.SQSMessage message : sqsEvent.getRecords()) {
            try {
                //process your message
            } catch (Exception e) {
                //Add failed message identifier to the batchItemFailures
 list
                batchItemFailures.add(new
 SQSBatchResponse.BatchItemFailure(message.getMessageId()));
            }
        }
        return new SQSBatchResponse(batchItemFailures);
    }
}
```

JavaScript

SDK untuk JavaScript (v3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch SQS dengan penggunaan JavaScript Lambda.

```
// Node.js 20.x Lambda runtime, AWS SDK for Javascript V3
export const handler = async (event, context) => {
    const batchItemFailures = [];
    for (const record of event.Records) {
        try {
            await processMessageAsync(record, context);
        } catch (error) {
            batchItemFailures.push({ itemIdentifier: record.messageId });
        }
    }
    return { batchItemFailures };
};

async function processMessageAsync(record, context) {
    if (record.body && record.body.includes("error")) {
        throw new Error("There is an error in the SQS Message.");
    }
    console.log(`Processed message: ${record.body}`);
}
```

Melaporkan kegagalan item batch SQS dengan penggunaan TypeScript Lambda.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { SQSEvent, SQSBatchResponse, Context, SQSBatchItemFailure,
 SQSRecord } from 'aws-lambda';
```

```
export const handler = async (event: SQSEvent, context: Context):
 Promise<SQSBatchResponse> => {
    const batchItemFailures: SQSBatchItemFailure[] = [];

    for (const record of event.Records) {
        try {
            await processMessageAsync(record);
        } catch (error) {
            batchItemFailures.push({ itemIdentifier: record.messageId });
        }
    }

    return {batchItemFailures: batchItemFailures};
};

async function processMessageAsync(record: SQSRecord): Promise<void> {
    if (record.body && record.body.includes("error")) {
        throw new Error('There is an error in the SQS Message.');
    }
    console.log(`Processed message ${record.body}`);
}
```

PHP

SDK untuk PHP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch SQS dengan Lambda menggunakan PHP.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

use Bref\Context\Context;
use Bref\Event\Sqs\SqsEvent;
```

```
use Bref\Event\Sqs\SqsHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler extends SqsHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handleSqs(SqsEvent $event, Context $context): void
    {
        $this->logger->info("Processing SQS records");
        $records = $event->getRecords();

        foreach ($records as $record) {
            try {
                // Assuming the SQS message is in JSON format
                $message = json_decode($record->getBody(), true);
                $this->logger->info(json_encode($message));
                // TODO: Implement your custom processing logic here
            } catch (Exception $e) {
                $this->logger->error($e->getMessage());
                // failed processing the record
                $this->markAsFailed($record);
            }
        }
        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords SQS
 records");
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch SQS dengan Lambda menggunakan Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0

def lambda_handler(event, context):
    if event:
        batch_item_failures = []
        sqs_batch_response = {}

        for record in event["Records"]:
            try:
                print(f"Processed message: {record['body']}")
            except Exception as e:
                batch_item_failures.append({"itemIdentifier":
 record['messageId']})

        sqs_batch_response["batchItemFailures"] = batch_item_failures
        return sqs_batch_response
```

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch SQS dengan Lambda menggunakan Ruby.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
require 'json'

def lambda_handler(event:, context:)
  if event
    batch_item_failures = []
    sqs_batch_response = {}

    event["Records"].each do |record|
      begin
        # process message
      rescue StandardError => e
        batch_item_failures << {"itemIdentifier" => record['messageId']}
      end
    end

    sqs_batch_response["batchItemFailures"] = batch_item_failures
    return sqs_batch_response
  end
end
```

Rust

SDK for Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch SQS dengan Lambda menggunakan Rust.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
```

```
use aws_lambda_events::{
    event::sqs::{SqsBatchResponse, SqsEvent},
    sqs::{BatchItemFailure, SqsMessage},
};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

async fn process_record(_: &SqsMessage) -> Result<(), Error> {
    Err(Error::from("Error processing message"))
}

async fn function_handler(event: LambdaEvent<SqsEvent>) ->
 Result<SqsBatchResponse, Error> {
    let mut batch_item_failures = Vec::new();
    for record in event.payload.records {
        match process_record(&record).await {
            Ok(_) => (),
            Err(_) => batch_item_failures.push(BatchItemFailure {
                item_identifier: record.message_id.unwrap(),
            }),
        }
    }

    Ok(SqsBatchResponse {
        batch_item_failures,
    })
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    run(service_fn(function_handler)).await
}
```

Jika peristiwa gagal tidak kembali ke antrian, lihat Bagaimana cara memecahkan masalah fungsi Lambda SQS? ReportBatchItemFailures di pusat AWS pengetahuan.

Status berhasil dan gagal

Lambda memperlakukan batch sebagai sukses total jika fungsi Anda mengembalikan salah satu dari berikut ini:

- Daftar `batchItemFailures` kosong

- Daftar `batchItemFailures` nol
- `EventResponse` kosong
- `EventResponse` nol

Lambda memperlakukan batch sebagai kegagalan total jika fungsi Anda mengembalikan salah satu dari berikut ini:

- Respons JSON yang tidak valid
- String `itemIdentifier` kosong
- `itemIdentifier` nol
- `itemIdentifier` dengan nama kunci yang buruk
- `itemIdentifier`Nilai dengan ID pesan yang tidak ada

CloudWatch metrik

Untuk menentukan apakah fungsi Anda melaporkan kegagalan item batch dengan benar, Anda dapat memantau metrik `ApproximateAgeOfOldestMessage` Amazon SQS `NumberOfMessagesDeleted` dan Amazon di Amazon. CloudWatch

- `NumberOfMessagesDeleted`melacak jumlah pesan yang dihapus dari antrian Anda. Jika ini turun ke 0, ini adalah tanda bahwa respons fungsi Anda tidak mengembalikan pesan yang gagal dengan benar.
- `ApproximateAgeOfOldestMessage`melacak berapa lama pesan tertua telah tinggal di antrian Anda. Peningkatan tajam dalam metrik ini dapat menunjukkan bahwa fungsi Anda tidak mengembalikan pesan yang gagal dengan benar.

Menggunakan Powertools untuk AWS Lambda prosesor batch

Utilitas prosesor batch dari Powertools untuk AWS Lambda secara otomatis menangani logika respons batch paral, mengurangi kompleksitas penerapan pelaporan kegagalan batch. Berikut adalah contoh menggunakan prosesor batch:

Python

> **Note**
>
> Untuk contoh lengkap dan petunjuk penyiapan, lihat dokumentasi prosesor batch.

Memproses pesan Amazon SQS dengan prosesor AWS Lambda batch.

```
import json
from aws_lambda_powertools import Logger
from aws_lambda_powertools.utilities.batch import BatchProcessor, EventType,
 process_partial_response
from aws_lambda_powertools.utilities.data_classes import SQSEvent
from aws_lambda_powertools.utilities.typing import LambdaContext

processor = BatchProcessor(event_type=EventType.SQS)
logger = Logger()

def record_handler(record):
    logger.info(record)
    # Your business logic here
    # Raise an exception to mark this record as failed

def lambda_handler(event, context: LambdaContext):
    return process_partial_response(
        event=event,
        record_handler=record_handler,
        processor=processor,
        context=context
    )
```

TypeScript

> **Note**
>
> Untuk contoh lengkap dan petunjuk penyiapan, lihat dokumentasi prosesor batch.

Memproses pesan Amazon SQS dengan prosesor AWS Lambda batch.

```
import { BatchProcessor, EventType, processPartialResponse } from '@aws-lambda-
powertools/batch';
import { Logger } from '@aws-lambda-powertools/logger';
import type { SQSEvent, Context } from 'aws-lambda';

const processor = new BatchProcessor(EventType.SQS);
const logger = new Logger();

const recordHandler = async (record: any): Promise<void> => {
    logger.info('Processing record', { record });
    // Your business logic here
    // Throw an error to mark this record as failed
};

export const handler = async (event: SQSEvent, context: Context) => {
    return processPartialResponse(event, recordHandler, processor, {
        context,
    });
};
```

Parameter Lambda untuk pemetaan sumber peristiwa Amazon SQS

Semua jenis sumber peristiwa Lambda berbagi operasi yang sama CreateEventSourceMappingdan UpdateEventSourceMappingAPI. Namun, hanya beberapa parameter yang berlaku untuk Amazon SQS.

Parameter	Diperlukan	Default	Catatan
BatchSize	T	10	Untuk antrian standar, maksimum 10.000. Untuk antrian FIFO, maksimum adalah 10.
Diaktifkan	T	true	none
EventSourceArn	Y	N/A	ARN dari aliran data atau konsumen aliran
FunctionName	Y	N/A	none

Parameter	Diperlukan	Default	Catatan
FilterCriteria	T	N/A	Kontrol peristiwa mana yang dikirim Lambda ke Fungsi Acara Anda
FunctionResponseTypes	T	N/A	Untuk membiarkan Fungsi Acara melaporkan kegagalan tertentu dalam batch, sertakan nilainya `ReportBatchItemFailures FunctionResponseTypes` . Untuk informasi selengkapnya, lihat Menerapkan tanggapan batch sebagian.
MaximumBatchingWindowInSeconds	T	0	Jendela batching tidak didukung untuk antrian FIFO

Parameter	Diperlukan	Default	Catatan
ProvisionedPollerConfig	T	N/A	Mengkonfigurasi jumlah minimum (2-200) dan maksimum (2-2000) poller acara khusus untuk pemetaan sumber peristiwa SQS. Setiap poller dapat menangani hingga 1 throughput dan 10 MB/sec pemanggilan bersamaan.
ScalingConfig	T	N/A	Mengkonfigurasi konkurensi maksimum untuk sumber peristiwa Amazon SQS

Menggunakan pemfilteran peristiwa dengan sumber peristiwa Amazon SQS

Anda dapat menggunakan pemfilteran peristiwa untuk mengontrol rekaman mana dari aliran atau antrian yang dikirim Lambda ke Fungsi Acara Anda. Untuk informasi umum tentang cara kerja penyaringan acara, lihatthe section called “Pemfilteran peristiwa”.

Bagian ini berfokus pada pemfilteran acara untuk sumber acara Amazon SQS.

> **Note**
>
> Pemetaan sumber peristiwa Amazon SQS hanya mendukung pemfilteran pada kunci. body

Topik

- Dasar-dasar pemfilteran acara Amazon SQS

Dasar-dasar pemfilteran acara Amazon SQS

Misalkan antrian Amazon SQS Anda berisi pesan dalam format JSON berikut.

```
{
    "RecordNumber": 1234,
    "TimeStamp": "yyyy-mm-ddThh:mm:ss",
    "RequestCode": "AAAA"
}
```

Contoh catatan untuk antrian ini akan terlihat sebagai berikut.

```
{
    "messageId": "059f36b4-87a3-44ab-83d2-661975830a7d",
    "receiptHandle": "AQEBwJnKyrHigUMZj6rYigCgxlaS3SLy0a...",
    "body": "{\n "RecordNumber": 1234,\n "TimeStamp": "yyyy-mm-ddThh:mm:ss",\n
 "RequestCode": "AAAA"\n}",
    "attributes": {
        "ApproximateReceiveCount": "1",
        "SentTimestamp": "1545082649183",
        "SenderId": "AIDAIENQZJOLO23YVJ4VO",
        "ApproximateFirstReceiveTimestamp": "1545082649185"
        },
    "messageAttributes": {},
    "md5OfBody": "e4e68fb7bd0e697a0ae8f1bb342846b3",
    "eventSource": "aws:sqs",
    "eventSourceARN": "arn:aws:sqs:us-west-2:123456789012:my-queue",
    "awsRegion": "us-west-2"
}
```

Untuk memfilter berdasarkan konten pesan Amazon SQS Anda, gunakan `body` kunci dalam catatan pesan Amazon SQS. Misalkan Anda hanya ingin memproses catatan tersebut di mana pesan Amazon SQS Anda adalah “BBBB.” `RequestCode` `FilterCriteria`Objeknya adalah sebagai berikut.

```
{
    "Filters": [
        {
            "Pattern": "{ \"body\" : { \"RequestCode\" : [ \"BBBB\" ] } }"
        }
    ]
}
```

Untuk kejelasan tambahan, berikut adalah nilai filter yang `Pattern` diperluas di JSON biasa.

```
{
    "body": {
        "RequestCode": [ "BBBB" ]
        }
}
```

Anda dapat menambahkan filter menggunakan konsol, AWS CLI atau AWS SAM templat.

Console

Untuk menambahkan filter ini menggunakan konsol, ikuti instruksi Melampirkan kriteria filter ke pemetaan sumber peristiwa (konsol) dan masukkan string berikut untuk kriteria Filter.

```
{ "body" : { "RequestCode" : [ "BBBB" ] } }
```

AWS CLI

Untuk membuat pemetaan sumber peristiwa baru dengan kriteria filter ini menggunakan AWS Command Line Interface (AWS CLI), jalankan perintah berikut.

```
aws lambda create-event-source-mapping \
    --function-name my-function \
    --event-source-arn arn:aws:sqs:us-east-2:123456789012:my-queue \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"body\" : { \"RequestCode\" :
 [ \"BBBB\" ] } }"}]}'
```

Untuk menambahkan kriteria filter ini ke pemetaan sumber peristiwa yang ada, jalankan perintah berikut.

```
aws lambda update-event-source-mapping \
    --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"body\" : { \"RequestCode\" :
 [ \"BBBB\" ] } }"}]}'
```

AWS SAM

Untuk menambahkan filter ini menggunakan AWS SAM, tambahkan cuplikan berikut ke template YAMB untuk sumber acara Anda.

```
FilterCriteria:
```

```
  Filters:
    - Pattern: '{ "body" : { "RequestCode" : [ "BBBB" ] } }'
```

Misalkan Anda ingin Fungsi Acara Anda memproses hanya catatan yang `RecordNumber` lebih besar dari 9999. `FilterCriteria`Objeknya adalah sebagai berikut.

```
{
    "Filters": [
        {
            "Pattern": "{ \"body\" : { \"RecordNumber\" : [ { \"numeric\": [ \">\",
 9999 ] } ] } }"
        }
    ]
}
```

Untuk kejelasan tambahan, berikut adalah nilai filter yang `Pattern` diperluas di JSON biasa.

```
{
    "body": {
        "RecordNumber": [
            {
                "numeric": [ ">", 9999 ]
            }
        ]
    }
}
```

Anda dapat menambahkan filter menggunakan konsol, AWS CLI atau AWS SAM templat.

Console

Untuk menambahkan filter ini menggunakan konsol, ikuti instruksi Melampirkan kriteria filter ke pemetaan sumber peristiwa (konsol) dan masukkan string berikut untuk kriteria Filter.

```
{ "body" : { "RecordNumber" : [ { "numeric": [ ">", 9999 ] } ] } }
```

AWS CLI

Untuk membuat pemetaan sumber peristiwa baru dengan kriteria filter ini menggunakan AWS Command Line Interface (AWS CLI), jalankan perintah berikut.

```
aws lambda create-event-source-mapping \
    --function-name my-function \
    --event-source-arn arn:aws:sqs:us-east-2:123456789012:my-queue \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"body\" : { \"RecordNumber\" :
 [ { \"numeric\": [ \">\", 9999 ] } ] } }"}]}'
```

Untuk menambahkan kriteria filter ini ke pemetaan sumber peristiwa yang ada, jalankan perintah berikut.

```
aws lambda update-event-source-mapping \
    --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"body\" : { \"RecordNumber\" :
 [ { \"numeric\": [ \">\", 9999 ] } ] } }"}]}'
```

AWS SAM

Untuk menambahkan filter ini menggunakan AWS SAM, tambahkan cuplikan berikut ke template YAMB untuk sumber acara Anda.

```
FilterCriteria:
  Filters:
    - Pattern: '{ "body" : { "RecordNumber" : [ { "numeric": [ ">", 9999 ] } ] } }'
```

Untuk Amazon SQS, badan pesan dapat berupa string apa pun. Namun, ini bisa menjadi masalah jika Anda `FilterCriteria` `body` berharap berada dalam format JSON yang valid. Skenario sebaliknya juga benar—jika isi pesan yang masuk dalam format JSON tetapi kriteria filter Anda diharapkan menjadi string biasa, ini dapat `body` menyebabkan perilaku yang tidak diinginkan.

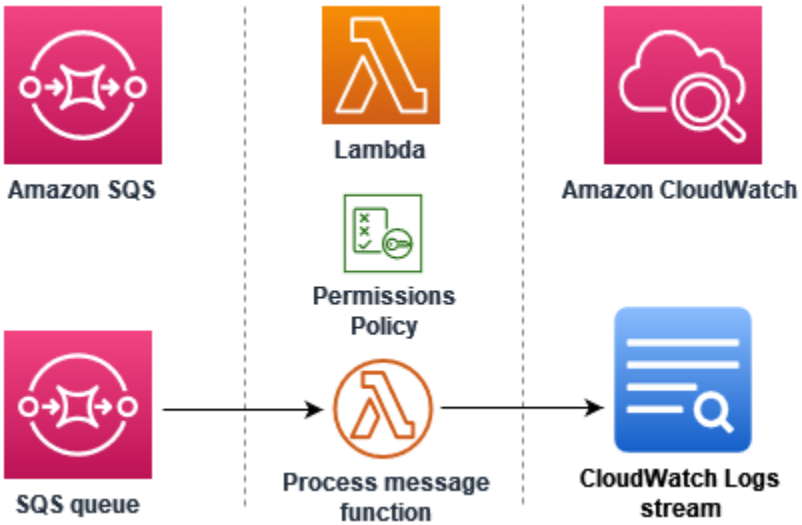
Untuk menghindari masalah ini, pastikan bahwa format isi dalam Anda `FilterCriteria` cocok dengan format yang diharapkan `body` dalam pesan yang Anda terima dari antrian Anda. Sebelum memfilter pesan Anda, Lambda secara otomatis mengevaluasi format badan pesan masuk dan pola filter Anda. `body` Jika ada ketidakcocokan, Lambda menjatuhkan pesan. Tabel berikut merangkum evaluasi ini:

Format pesan **body** masuk	**body**Format pola filter	Tindakan yang dihasilkan
Tali polos	Tali polos	Filter Lambda berdasarkan kriteria filter Anda.

Format pesan **body** masuk	**body**Format pola filter	Tindakan yang dihasilkan
Tali polos	Tidak ada pola filter untuk properti data	Filter Lambda (hanya pada properti metadata lainnya) berdasarkan kriteria filter Anda.
Tali polos	JSON yang valid	Lambda menjatuhkan pesan.
JSON yang valid	Tali polos	Lambda menjatuhkan pesan.
JSON yang valid	Tidak ada pola filter untuk properti data	Filter Lambda (hanya pada properti metadata lainnya) berdasarkan kriteria filter Anda.
JSON yang valid	JSON yang valid	Filter Lambda berdasarkan kriteria filter Anda.

Tutorial: Menggunakan Lambda dengan Amazon SQS

Dalam tutorial ini, Anda membuat Lambda Event Function yang menggunakan pesan dari antrian Amazon Simple Queue Service (Amazon SQS). Fungsi Peristiwa Lambda berjalan setiap kali pesan baru ditambahkan ke antrian. Fungsi ini menulis pesan ke aliran Amazon CloudWatch Logs. Diagram berikut menunjukkan sumber daya AWS yang Anda gunakan untuk menyelesaikan tutorial.



Untuk menyelesaikan tutorial ini, Anda melakukan langkah-langkah berikut:

1. Buat Fungsi Peristiwa Lambda yang menulis pesan ke CloudWatch Log.
2. Membuat antrean Amazon SQS.
3. Buat pemetaan sumber acara Lambda. Pemetaan sumber peristiwa membaca antrean Amazon SQS dan memanggil Fungsi Peristiwa Lambda Anda saat pesan baru ditambahkan.
4. Uji penyiapan dengan menambahkan pesan ke antrian Anda dan pantau hasilnya di CloudWatch Log.

Prasyarat

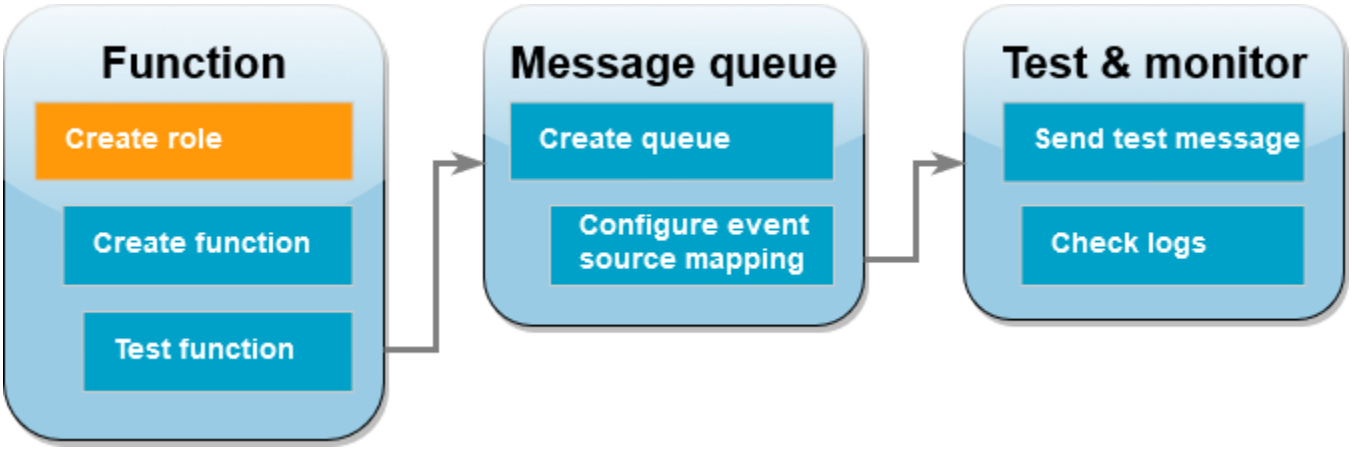
Instal AWS Command Line Interface

Jika Anda belum menginstal AWS Command Line Interface, ikuti langkah-langkah di Menginstal atau memperbarui versi terbaru AWS CLI untuk menginstalnya.

Tutorial ini membutuhkan terminal baris perintah atau shell untuk menjalankan perintah. Di Linux dan macOS, gunakan shell dan manajer paket pilihan Anda.

> **Note**
>
> Di Windows, beberapa perintah Bash CLI yang biasa Anda gunakan dengan Lambda (`zip`seperti) tidak didukung oleh terminal bawaan sistem operasi. Untuk mendapatkan Windows-integrated versi Ubuntu dan Bash, instal Subsistem Windows untuk Linux.

Buat peran eksekusi



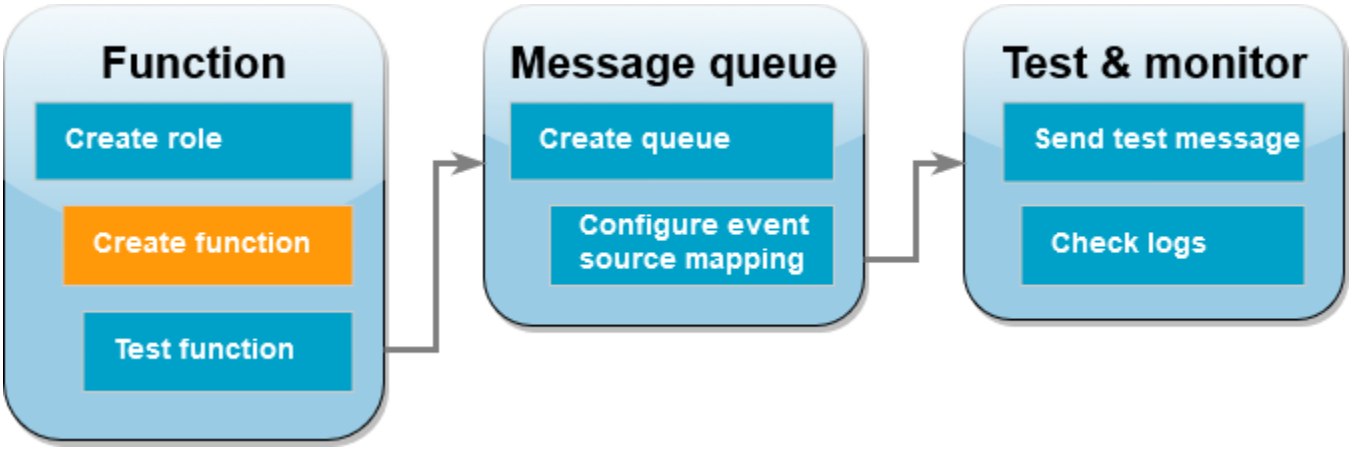
Peran eksekusi adalah peran AWS Identity and Access Management (IAM) yang memberikan izin Fungsi Peristiwa Lambda untuk mengakses dan sumber daya. Layanan AWS
Untuk mengizinkan fungsi Anda membaca item dari Amazon SQS, lampirkan kebijakan AWSLambdaSQSQueueExecutionRoleizin.

Untuk membuat peran eksekusi dan melampirkan kebijakan izin Amazon SQS

1. Buka halaman Peran dari konsol IAM.
2. Pilih Buat peran.
3. Untuk jenis entitas Tepercaya, pilih AWS layanan.
4. Untuk kasus penggunaan, pilih Lambda.
5. Pilih Berikutnya.
6. Di kotak pencarian Kebijakan izin, masukkan**`AWSLambdaSQSQueueExecutionRole`**.
7. Pilih AWSLambdaSQSQueueExecutionRolekebijakan, lalu pilih Berikutnya.
8. Di bawah Rincian peran, untuk nama Peran**`lambda-sqs-role`**, masukkan, lalu pilih Buat peran.

Setelah pembuatan peran, catat Nama Sumber Daya Amazon (ARN) peran eksekusi Anda. Anda akan membutuhkannya di langkah selanjutnya.

Buat Fungsi Acara



Buat Fungsi Peristiwa Lambda yang memproses pesan Amazon SQS Anda. Kode fungsi mencatat isi pesan Amazon SQS ke CloudWatch Log.

Tutorial ini menggunakan runtime Node.js 24, tetapi kami juga menyediakan kode contoh dalam bahasa runtime lainnya. Anda dapat memilih tab di kotak berikut untuk melihat kode runtime yang

Anda minati. JavaScript Kode yang akan Anda gunakan dalam langkah ini adalah pada contoh pertama yang ditunjukkan di JavaScripttab.

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara SQS dengan Lambda menggunakan.NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using Amazon.Lambda.Core;
using Amazon.Lambda.SQSEvents;


// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri

namespace SqsIntegrationSampleCode
{
    public async Task FunctionHandler(SQSEvent evnt, ILambdaContext context)
    {
        foreach (var message in evnt.Records)
        {
            await ProcessMessageAsync(message, context);
        }

        context.Logger.LogInformation("done");
    }

    private async Task ProcessMessageAsync(SQSEvent.SQSMessage message,
 ILambdaContext context)
    {
        try
```

```
        {
            context.Logger.LogInformation($"Processed message {message.Body}");

            // TODO: Do interesting work based on the new message
            await Task.CompletedTask;
        }
        catch (Exception e)
        {
            //You can use Dead Letter Queue to handle failures. By configuring a
 Lambda DLQ.
            context.Logger.LogError($"An error occurred");
            throw;
        }

    }
}
```

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara SQS dengan Lambda menggunakan Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package integration_sqs_to_lambda

import (
 "fmt"
 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

func handler(event events.SQSEvent) error {
 for _, record := range event.Records {
```

```
  err := processMessage(record)
  if err != nil {
   return err
  }
 }
 fmt.Println("done")
 return nil
}

func processMessage(record events.SQSMessage) error {
 fmt.Printf("Processed message %s\n", record.Body)
 // TODO: Do interesting work based on the new message
 return nil
}

func main() {
 lambda.Start(handler)
}
```

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara SQS dengan Lambda menggunakan Java.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.SQSEvent;
import com.amazonaws.services.lambda.runtime.events.SQSEvent.SQSMessage;

public class Function implements RequestHandler<SQSEvent, Void> {
    @Override
    public Void handleRequest(SQSEvent sqsEvent, Context context) {
```

```
        for (SQSMessage msg : sqsEvent.getRecords()) {
            processMessage(msg, context);
        }
        context.getLogger().log("done");
        return null;
    }

    private void processMessage(SQSMessage msg, Context context) {
        try {
            context.getLogger().log("Processed message " + msg.getBody());

            // TODO: Do interesting work based on the new message

        } catch (Exception e) {
            context.getLogger().log("An error occurred");
            throw e;
        }

    }
}
```

JavaScript

SDK untuk JavaScript (v3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara SQS dengan JavaScript Lambda menggunakan.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const message of event.Records) {
    await processMessageAsync(message);
  }
  console.info("done");
};
```

```
async function processMessageAsync(message) {
  try {
    console.log(`Processed message ${message.body}`);
    // TODO: Do interesting work based on the new message
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

Mengkonsumsi acara SQS dengan TypeScript Lambda menggunakan.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { SQSEvent, Context, SQSHandler, SQSRecord } from "aws-lambda";

export const functionHandler: SQSHandler = async (
  event: SQSEvent,
  context: Context
): Promise<void> => {
  for (const message of event.Records) {
    await processMessageAsync(message);
  }
  console.info("done");
};

async function processMessageAsync(message: SQSRecord): Promise<any> {
  try {
    console.log(`Processed message ${message.body}`);
    // TODO: Do interesting work based on the new message
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

PHP

SDK untuk PHP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara SQS dengan Lambda menggunakan PHP.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\InvalidLambdaEvent;
use Bref\Event\Sqs\SqsEvent;
use Bref\Event\Sqs\SqsHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler extends SqsHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws InvalidLambdaEvent
     */
    public function handleSqs(SqsEvent $event, Context $context): void
    {
        foreach ($event->getRecords() as $record) {
            $body = $record->getBody();
            // TODO: Do interesting work based on the new message
        }
```

```
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara SQS dengan Lambda menggunakan Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def lambda_handler(event, context):
    for message in event['Records']:
        process_message(message)
    print("done")

def process_message(message):
    try:
        print(f"Processed message {message['body']}")
        # TODO: Do interesting work based on the new message
    except Exception as err:
        print("An error occurred")
        raise err
```

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara SQS dengan Lambda menggunakan Ruby.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def lambda_handler(event:, context:)
  event['Records'].each do |message|
    process_message(message)
  end
  puts "done"
end

def process_message(message)
  begin
    puts "Processed message #{message['body']}"
    # TODO: Do interesting work based on the new message
  rescue StandardError => err
    puts "An error occurred"
    raise err
  end
end
```

Rust

SDK for Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara SQS dengan Lambda menggunakan Rust.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::event::sqs::SqsEvent;
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

async fn function_handler(event: LambdaEvent<SqsEvent>) -> Result<(), Error> {
    event.payload.records.iter().for_each(|record| {
        // process the record
        tracing::info!("Message body: {}",
 record.body.as_deref().unwrap_or_default())
    });

    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        // disable printing the name of the module in every log line.
        .with_target(false)
        // disabling time is handy because CloudWatch will add the ingestion
 time.
        .without_time()
        .init();

    run(service_fn(function_handler)).await
}
```

Untuk membuat Fungsi Acara Node.js Lambda

1. Buat direktori untuk proyek, dan kemudian beralih ke direktori itu.

```
mkdir sqs-tutorial
cd sqs-tutorial
```

2. Salin JavaScript kode sampel ke file baru bernama`index.js`.
3. Buat paket penyebaran menggunakan `zip` perintah berikut.

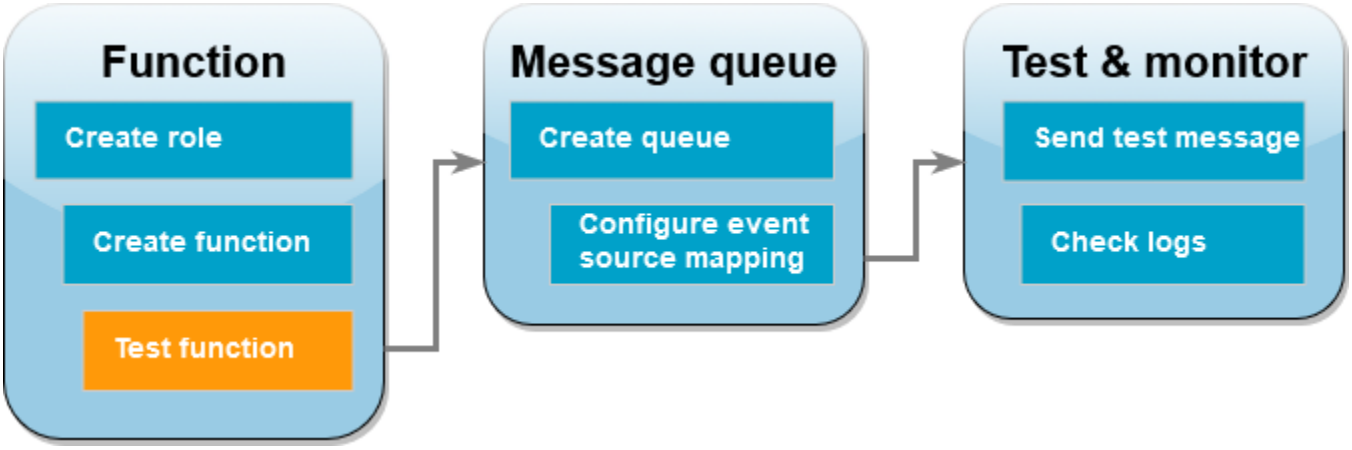
```
zip function.zip index.js
```

4. Buat Fungsi Peristiwa Lambda menggunakan perintah AWS CLI create-function. Untuk parameter `role`, masukkan ARN peran eksekusi yang Anda buat sebelumnya.

> **Note**
>
> Fungsi Peristiwa Lambda dan antrean Amazon SQS harus sama. Wilayah AWS

```
aws lambda create-function --function-name ProcessSQSRecord \
--zip-file fileb://function.zip --handler index.handler --runtime nodejs24.x \
--role arn:aws:iam::111122223333:role/lambda-sqs-role
```

Uji Fungsi Acara



Memanggil Fungsi Peristiwa Lambda Anda secara manual menggunakan `invoke` AWS CLI perintah dan contoh peristiwa Amazon SQS.

Untuk memanggil Fungsi Peristiwa Lambda dengan contoh peristiwa

1. Simpan JSON berikut sebagai file bernama`input.json`. JSON ini mensimulasikan peristiwa yang mungkin dikirimkan Amazon SQS ke Fungsi Peristiwa Lambda Anda, yang berisi pesan aktual dari `"body"` antrian. Dalam contoh ini, pesannya adalah`"test"`.

   Example Acara Amazon SQS

   Ini adalah acara pengujian—Anda tidak perlu mengubah pesan atau nomor akun.

```
{
    "Records": [
        {
            "messageId": "059f36b4-87a3-44ab-83d2-661975830a7d",
            "receiptHandle": "AQEBwJnKyrHigUMZj6rYigCgxlaS3SLy0a...",
            "body": "test",
            "attributes": {
                "ApproximateReceiveCount": "1",
                "SentTimestamp": "1545082649183",
                "SenderId": "AIDAIENQZJOLO23YVJ4VO",
                "ApproximateFirstReceiveTimestamp": "1545082649185"
            },
            "messageAttributes": {},
            "md5OfBody": "098f6bcd4621d373cade4e832627b4f6",
            "eventSource": "aws:sqs",
            "eventSourceARN": "arn:aws:sqs:us-east-1:111122223333:my-queue",
            "awsRegion": "us-east-1"
        }
    ]
}
```

2. Jalankan AWS CLI perintah pemanggilan berikut. Perintah ini mengembalikan CloudWatch log dalam respon. Untuk informasi selengkapnya tentang mendapatkan log kembali, lihat Akses log dengan AWS CLI.

```
aws lambda invoke --function-name ProcessSQSRecord --payload file://input.json out
 --log-type Tail \
--query 'LogResult' --output text --cli-binary-format raw-in-base64-out | base64 --
decode
```

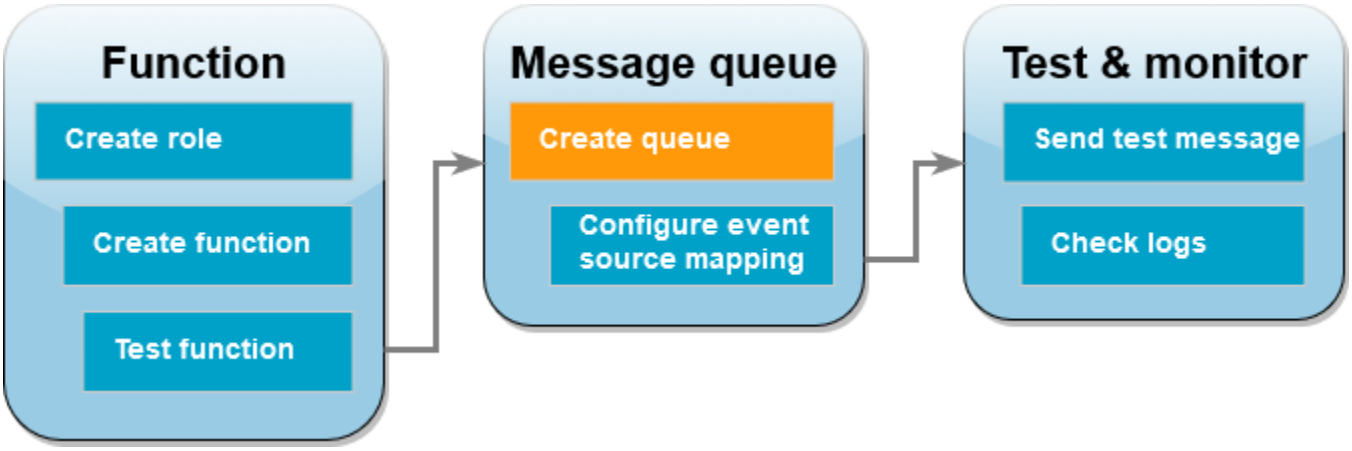
cli-binary-formatOpsi ini diperlukan jika Anda menggunakan AWS CLI versi 2. Untuk menjadikan ini pengaturan default, jalankan`aws configure set cli-binary-format raw-in-base64-out`. Untuk informasi selengkapnya, lihat opsi baris perintah global yang AWS CLI didukung di Panduan AWS Command Line Interface Pengguna untuk Versi 2.

3. Temukan `INFO` log di respons. Di sinilah Lambda Event Function mencatat isi pesan. Anda akan melihat log yang terlihat seperti ini:

```
2023-09-11T22:45:04.271Z 348529ce-2211-4222-9099-59d07d837b60 INFO Processed
 message test
```

```
2023-09-11T22:45:04.288Z 348529ce-2211-4222-9099-59d07d837b60 INFO done
```

Buat antrean Amazon SQS



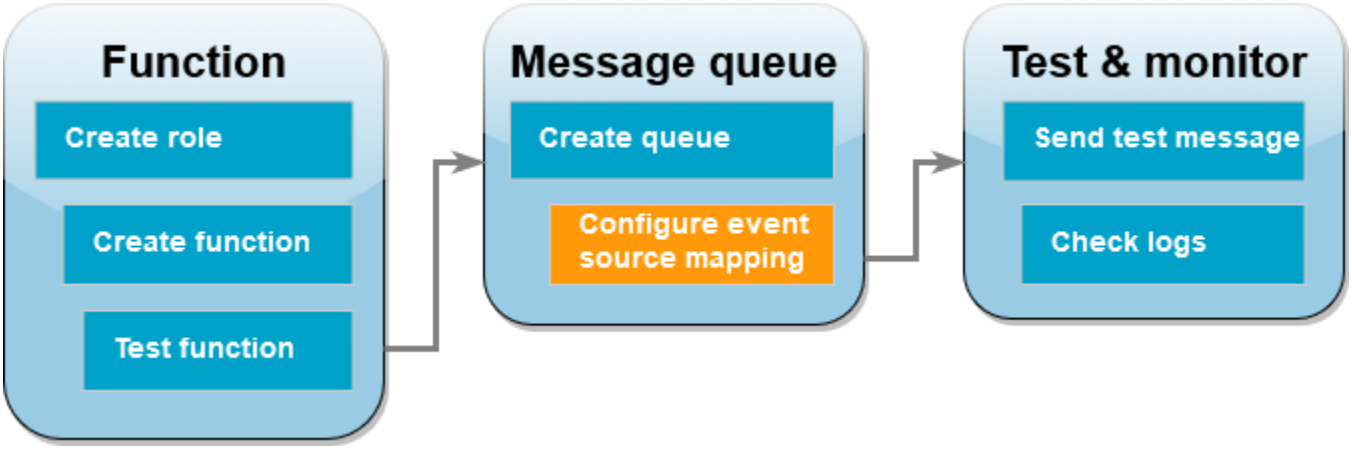
Buat antrean Amazon SQS yang dapat digunakan oleh Fungsi Peristiwa Lambda sebagai sumber peristiwa. Fungsi Peristiwa Lambda dan antrean Amazon SQS harus sama. Wilayah AWS

Untuk membuat antrean

1. Buka konsol Amazon SQS.
2. Pilih Buat antrean.
3. Masukkan nama untuk antrian. Biarkan semua opsi lain di pengaturan default.
4. Pilih Buat antrean.

Setelah membuat antrian, catat ARN-nya. Anda memerlukan ini di langkah berikutnya ketika Anda mengaitkan antrian dengan Fungsi Acara Lambda Anda.

Konfigurasikan sumber kejadian



Hubungkan antrean Amazon SQS ke Fungsi Peristiwa Lambda Anda dengan membuat pemetaan sumber peristiwa. Pemetaan sumber peristiwa membaca antrean Amazon SQS dan memanggil Fungsi Peristiwa Lambda Anda saat pesan baru ditambahkan.

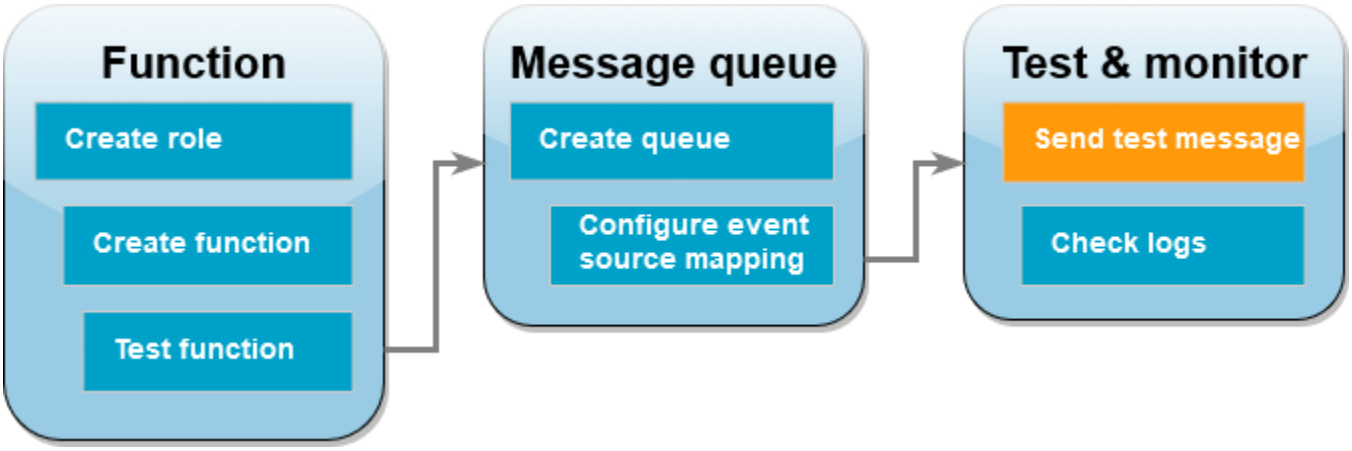
Untuk membuat pemetaan antara antrian Amazon SQS dan Fungsi Peristiwa Lambda, gunakan perintah create-event-source-mapping. AWS CLI Contoh:

```
aws lambda create-event-source-mapping --function-name ProcessSQSRecord  --batch-size
 10 \
--event-source-arn arn:aws:sqs:us-east-1:111122223333:my-queue
```

Untuk mendapatkan daftar pemetaan sumber acara Anda, gunakan perintah list-event-source-mappings. Contoh:

```
aws lambda list-event-source-mappings --function-name ProcessSQSRecord
```

Mengirim pesan pengujian

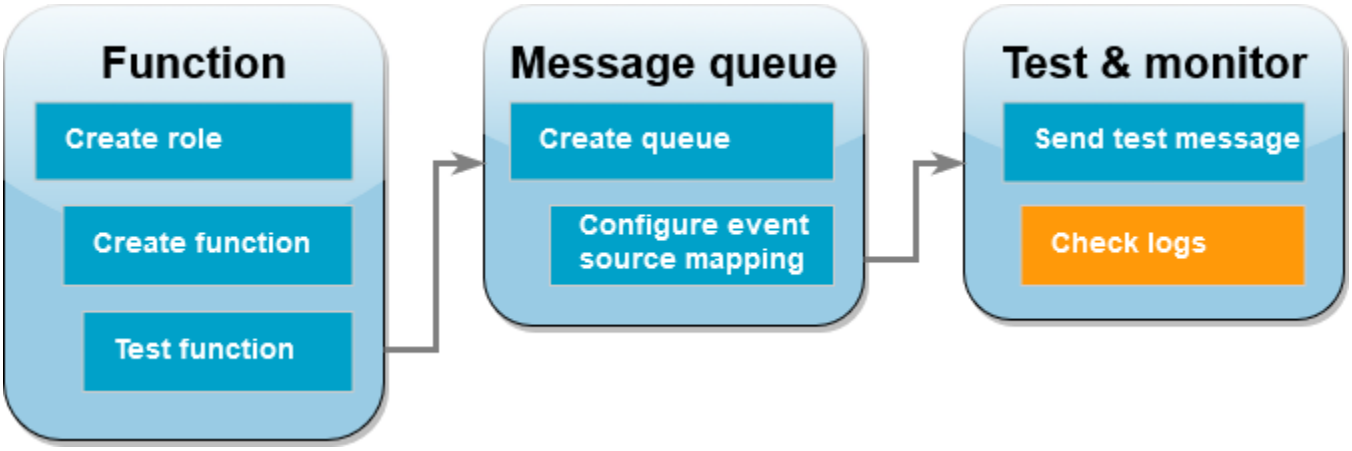


Untuk mengirim pesan Amazon SQS ke Fungsi Peristiwa Lambda

1. Buka konsol Amazon SQS.
2. Pilih antrian yang Anda buat sebelumnya.
3. Pilih Kirim dan terima pesan.
4. Di bawah Badan pesan, masukkan pesan pengujian, seperti “ini adalah pesan pengujian.”
5. Pilih Kirim pesan.

Lambda melakukan polling antrian untuk pembaruan. Ketika ada pesan baru, Lambda memanggil fungsi Anda dengan data peristiwa baru ini dari antrian. Jika fungsi handler kembali tanpa pengecualian, Lambda menganggap pesan berhasil diproses dan mulai membaca pesan baru dalam antrian. Setelah berhasil memproses pesan, Lambda secara otomatis menghapusnya dari antrian. Jika handler melempar pengecualian, Lambda menganggap kumpulan pesan tidak berhasil diproses, dan Lambda memanggil fungsi dengan kumpulan pesan yang sama.

Periksa CloudWatch log



Untuk mengonfirmasi bahwa Fungsi Acara memproses pesan

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih fungsi ProcessSQSRecord.
3. Pilih Monitor.
4. Pilih Lihat CloudWatch log.
5. Di CloudWatch konsol, pilih aliran Log untuk fungsi tersebut.
6. Temukan `INFO` log. Di sinilah Lambda Event Function mencatat isi pesan. Anda akan melihat pesan yang Anda kirim dari antrian Amazon SQS. Contoh:

```
2023-09-11T22:49:12.730Z b0c41e9c-0556-5a8b-af83-43e59efeec71 INFO Processed
 message this is a test message.
```

Bersihkan sumber daya Anda

Sekarang Anda dapat menghapus sumber daya yang Anda buat untuk tutorial ini, kecuali Anda ingin mempertahankannya. Dengan menghapus AWS sumber daya yang tidak lagi Anda gunakan, Anda mencegah tagihan yang tidak perlu ke Anda Akun AWS.

Untuk menghapus peran eksekusi

1. Buka halaman Peran dari konsol IAM.
2. Pilih peran eksekusi yang Anda buat.
3. Pilih Hapus.
4. Masukkan nama peran di bidang input teks dan pilih Hapus.

Untuk menghapus Fungsi Acara Lambda

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih Fungsi Acara yang Anda buat.
3. Pilih Tindakan, Hapus.
4. Ketik **`confirm`** kolom input teks dan pilih Hapus.

Untuk menghapus antrean Amazon SQS

1. Masuk ke Konsol Manajemen AWS dan buka konsol Amazon SQS di. https://console.aws.amazon.com/sqs/
2. Pilih antrean yang Anda buat.
3. Pilih Hapus.
4. Masukkan **`confirm`** di bidang input teks.
5. Pilih Hapus.

Tutorial: Menggunakan antrian Amazon SQS lintas akun sebagai sumber acara

Dalam tutorial ini, Anda membuat Lambda Event Function yang menggunakan pesan dari antrian Amazon Simple Queue Service (Amazon SQS) di akun yang berbeda. AWS Tutorial ini melibatkan dua AWS akun: Akun A mengacu pada akun yang berisi Fungsi Peristiwa Lambda Anda, dan Akun B mengacu pada akun yang berisi antrian Amazon SQS.

Prasyarat

Instal AWS Command Line Interface

Jika Anda belum menginstal AWS Command Line Interface, ikuti langkah-langkah di Menginstal atau memperbarui versi terbaru AWS CLI untuk menginstalnya.

Tutorial ini membutuhkan terminal baris perintah atau shell untuk menjalankan perintah. Di Linux dan macOS, gunakan shell dan manajer paket pilihan Anda.

> **Note**
>
> Di Windows, beberapa perintah Bash CLI yang biasa Anda gunakan dengan Lambda (`zip`seperti) tidak didukung oleh terminal bawaan sistem operasi. Untuk mendapatkan Windows-integrated versi Ubuntu dan Bash, instal Subsistem Windows untuk Linux.

Buat peran eksekusi (Akun A)

Di Akun A, buat peran eksekusi yang memberikan izin fungsi Anda untuk mengakses AWS sumber daya yang diperlukan.

Untuk membuat peran eksekusi

1. Buka halaman Peran di konsol AWS Identity and Access Management (IAM).
2. Pilih Buat peran.
3. Buat peran dengan properti berikut.
   - Entitas tepercaya - AWS Lambda
   - Izin – AWSLambdaSQSQueueExecutionRole
   - Nama peran – **`cross-account-lambda-sqs-role`**

AWSLambdaSQSQueueExecutionRoleKebijakan ini memiliki izin yang diperlukan fungsi untuk membaca item dari Amazon SQS dan untuk menulis log ke Amazon CloudWatch Logs.

Buat Fungsi Acara (Akun A)

Di Akun A, buat Fungsi Peristiwa Lambda yang memproses pesan Amazon SQS Anda. Fungsi Acara Lambda dan antrean Amazon SQS harus sama. Wilayah AWS

Contoh Node.js kode berikut menulis setiap pesan ke log in CloudWatch Log.

Example index.mjs

```
export const handler = async function(event, context) {
  event.Records.forEach(record => {
    const { body } = record;
    console.log(body);
  });
  return {};
}
```

Untuk membuat Fungsi Acara

> **Note**
>
> Mengikuti langkah-langkah ini menciptakan Node.js fungsi. Untuk bahasa lain, langkah-langkahnya serupa, tetapi beberapa detailnya berbeda.

1. Simpan contoh kode sebagai file bernama`index.mjs`.
2. Buat paket deployment.

   ```
   zip function.zip index.mjs
   ```

3. Buat fungsi menggunakan perintah `create-function` AWS Command Line Interface (AWS CLI). Ganti `arn:aws:iam::111122223333:role/cross-account-lambda-sqs-role` dengan ARN dari peran eksekusi yang Anda buat sebelumnya.

   ```
   aws lambda create-function --function-name CrossAccountSQSExample \
   --zip-file fileb://function.zip --handler index.handler --runtime nodejs24.x \
   --role arn:aws:iam::111122223333:role/cross-account-lambda-sqs-role
   ```

Uji Fungsi Acara (Akun A)

Di Akun A, uji Fungsi Peristiwa Lambda Anda secara manual menggunakan `invoke` AWS CLI perintah dan contoh peristiwa Amazon SQS.

Jika handler kembali normal tanpa pengecualian, Lambda menganggap pesan berhasil diproses dan mulai membaca pesan baru dalam antrian. Setelah berhasil memproses pesan, Lambda secara otomatis menghapusnya dari antrian. Jika handler melempar pengecualian, Lambda menganggap kumpulan pesan tidak berhasil diproses, dan Lambda memanggil fungsi dengan kumpulan pesan yang sama.

1. Simpan JSON berikut sebagai file bernama`input.txt`.

```
{
    "Records": [
        {
            "messageId": "059f36b4-87a3-44ab-83d2-661975830a7d",
            "receiptHandle": "AQEBwJnKyrHigUMZj6rYigCgxlaS3SLy0a...",
            "body": "test",
            "attributes": {
                "ApproximateReceiveCount": "1",
                "SentTimestamp": "1545082649183",
                "SenderId": "AIDAIENQZJOLO23YVJ4VO",
                "ApproximateFirstReceiveTimestamp": "1545082649185"
            },
            "messageAttributes": {},
            "md5OfBody": "098f6bcd4621d373cade4e832627b4f6",
            "eventSource": "aws:sqs",
            "eventSourceARN": "arn:aws:sqs:us-east-1:111122223333:example-queue",
            "awsRegion": "us-east-1"
        }
    ]
}
```

   JSON sebelumnya mensimulasikan peristiwa yang mungkin dikirimkan Amazon SQS ke Fungsi Peristiwa Lambda Anda, yang berisi pesan aktual dari antrian. `"body"`

2. Jalankan perintah `invoke` AWS CLI berikut.

```
aws lambda invoke --function-name CrossAccountSQSExample \
--cli-binary-format raw-in-base64-out \
--payload file://input.txt outputfile.txt
```

cli-binary-formatOpsi ini diperlukan jika Anda menggunakan AWS CLI versi 2. Untuk menjadikan ini pengaturan default, jalankan`aws configure set cli-binary-format raw-in-base64-out`. Untuk informasi selengkapnya, lihat opsi baris perintah global yang AWS CLI didukung di Panduan AWS Command Line Interface Pengguna untuk Versi 2.

3. Verifikasikan output dalam `outputfile.txt` file.

Buat antrian Amazon SQS (Akun B)

Di Akun B, buat antrean Amazon SQS yang dapat digunakan oleh Fungsi Peristiwa Lambda di Akun A sebagai sumber peristiwa. Fungsi Acara Lambda dan antrean Amazon SQS harus sama. Wilayah AWS

Untuk membuat antrean

1. Buka konsol Amazon SQS.
2. Pilih Buat antrean.
3. Buat antrian dengan properti berikut.
   - Jenis - Standar
   - Nama – LambdaCrossAccountQueue
   - Konfigurasi - Pertahankan pengaturan default.
   - Kebijakan akses — Pilih Lanjutan. Tempel dalam kebijakan JSON berikut. Ganti nilai-nilai berikut:
     - `111122223333`: Akun AWS ID untuk Akun A
     - `444455556666`: Akun AWS ID untuk Akun B

     JSON

```
{
    "Version":"2012-10-17",
    "Id": "Queue1_Policy_UUID",
    "Statement": [
        {
            "Sid": "Queue1_AllActions",
            "Effect": "Allow",
            "Principal": {
                "AWS": [
```

```
                    "arn:aws:iam::111122223333:role/cross-account-lambda-
sqs-role"
                ]
            },
            "Action": "sqs:*",
            "Resource": "arn:aws:sqs:us-
east-1:444455556666:LambdaCrossAccountQueue"
        }
    ]
}
```

Kebijakan ini memberikan peran eksekusi Lambda dalam izin Akun A untuk menggunakan pesan dari antrean Amazon SQS ini.

4. Setelah membuat antrian, catat Nama Sumber Daya Amazon (ARN). Anda memerlukan ini di langkah berikutnya ketika Anda mengaitkan antrian dengan Fungsi Acara Lambda Anda.

Konfigurasikan sumber acara (Akun A)

Di Akun A, buat pemetaan sumber peristiwa antara antrian Amazon SQS di Akun B dan Fungsi Peristiwa Lambda Anda dengan menjalankan perintah berikut. `create-event-source-mapping` AWS CLI Ganti `arn:aws:sqs:us-east-1:444455556666:LambdaCrossAccountQueue` dengan ARN antrean Amazon SQS yang Anda buat pada langkah sebelumnya.

```
aws lambda create-event-source-mapping --function-name CrossAccountSQSExample --batch-
size 10 \
--event-source-arn arn:aws:sqs:us-east-1:444455556666:LambdaCrossAccountQueue
```

Untuk mendapatkan daftar pemetaan sumber acara Anda, jalankan perintah berikut.

```
aws lambda list-event-source-mappings --function-name CrossAccountSQSExample \
--event-source-arn arn:aws:sqs:us-east-1:444455556666:LambdaCrossAccountQueue
```

Uji penyiapan

Anda sekarang dapat menguji pengaturan sebagai berikut:

1. Di Akun B, buka konsol Amazon SQS.
2. Pilih LambdaCrossAccountQueue, yang Anda buat sebelumnya.
3. Pilih Kirim dan terima pesan.

4. Di bawah Badan pesan, masukkan pesan pengujian.
5. Pilih Kirim pesan.

Fungsi Acara Lambda Anda di Akun A harus menerima pesan. Lambda terus melakukan polling antrian untuk pembaruan. Ketika ada pesan baru, Lambda memanggil fungsi Anda dengan data peristiwa baru ini dari antrian. Fungsi Anda berjalan dan membuat log di Amazon CloudWatch. Anda dapat melihat log di CloudWatch konsol.

Bersihkan sumber daya Anda

Sekarang Anda dapat menghapus sumber daya yang Anda buat untuk tutorial ini, kecuali Anda ingin mempertahankannya. Dengan menghapus AWS sumber daya yang tidak lagi Anda gunakan, Anda mencegah tagihan yang tidak perlu ke Anda Akun AWS.

Di Akun A, bersihkan peran eksekusi dan Fungsi Acara Lambda Anda.

Untuk menghapus peran eksekusi

1. Buka halaman Peran dari konsol IAM.
2. Pilih peran eksekusi yang Anda buat.
3. Pilih Hapus.
4. Masukkan nama peran di bidang input teks dan pilih Hapus.

Untuk menghapus Fungsi Acara Lambda

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih Fungsi Acara yang Anda buat.
3. Pilih Tindakan, Hapus.
4. Ketik **`confirm`** kolom input teks dan pilih Hapus.

Di Akun B, bersihkan antrian Amazon SQS.

Untuk menghapus antrean Amazon SQS

1. Masuk ke Konsol Manajemen AWS dan buka konsol Amazon SQS di. https://console.aws.amazon.com/sqs/
2. Pilih antrean yang Anda buat.

3. Pilih Hapus.
4. Masukkan **`confirm`** di bidang input teks.
5. Pilih Hapus.

Mengatur Fungsi Acara Lambda dengan Step Functions

AWS Step Functions menyediakan orkestrasi alur kerja visual untuk mengoordinasikan Fungsi Peristiwa Lambda dengan layanan lain. AWS Dengan integrasi asli ke 220+ AWS layanan dan infrastruktur tanpa pemeliharaan yang dikelola sepenuhnya, Step Functions sangat ideal ketika Anda membutuhkan desain alur kerja visual dan integrasi layanan yang dikelola sepenuhnya.

Untuk orkestrasi menggunakan bahasa pemrograman standar dalam Lambda di mana logika alur kerja hidup bersama logika bisnis, pertimbangkan fungsi Lambda yang tahan lama. Untuk bantuan memilih di antara opsi-opsi ini, lihat Fungsi tahan lama atau Step Functions.

Misalnya, memproses pesanan mungkin memerlukan validasi detail pesanan, memeriksa tingkat inventaris, memproses pembayaran, dan membuat faktur. Tulis Fungsi Peristiwa Lambda terpisah untuk setiap tugas dan gunakan Step Functions untuk mengelola alur kerja. Step Functions mengkoordinasikan aliran data antara fungsi Anda dan menangani kesalahan pada setiap langkah. Pemisahan ini membuat alur kerja Anda lebih mudah untuk divisualisasikan, dimodifikasi, dan dipelihara saat mereka tumbuh lebih kompleks.

Kapan menggunakan Step Functions dengan Lambda

Skenario berikut adalah contoh yang baik ketika Step Functions sangat cocok untuk mengatur aplikasi Lambda-based .

- Pemrosesan berurutan
- Penanganan kesalahan yang kompleks
- Alur kerja bersyarat dan persetujuan manusia
- Pemrosesan paralel

Pemrosesan berurutan

Pemrosesan sekuensial adalah ketika satu tugas harus diselesaikan sebelum tugas berikutnya dapat dimulai. Misalnya, dalam sistem pemrosesan pesanan, pemrosesan pembayaran tidak dapat dimulai sampai validasi pesanan selesai, dan pembuatan faktur harus menunggu konfirmasi pembayaran. Tulis Fungsi Peristiwa Lambda terpisah untuk setiap tugas dan gunakan Step Functions untuk mengelola urutan dan menangani aliran data antar fungsi.

Anti-pattern contoh

Satu Fungsi Peristiwa Lambda mengelola seluruh alur kerja pemrosesan pesanan dengan:

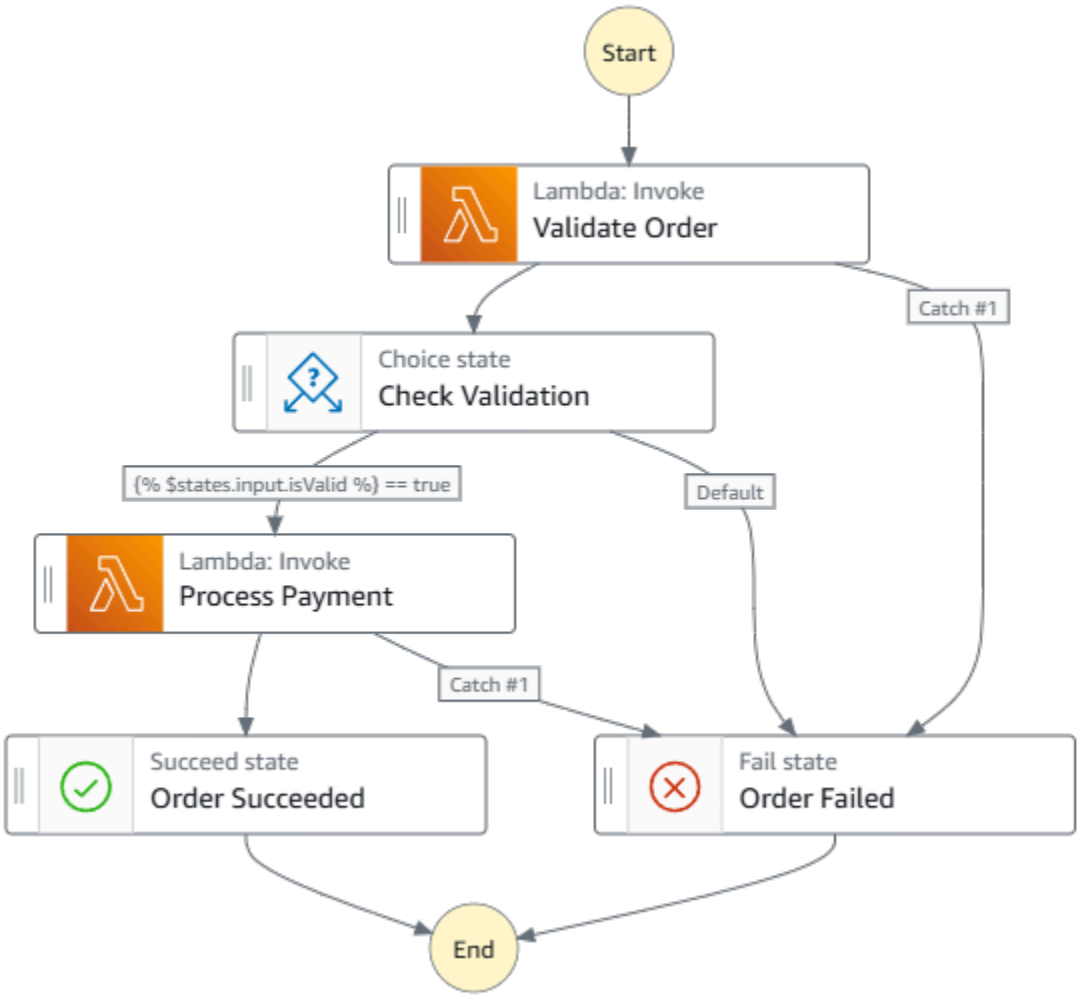
- Memanggil Fungsi Acara Lambda lainnya secara berurutan
- Mengurai dan memvalidasi tanggapan dari setiap fungsi
- Menerapkan penanganan kesalahan dan logika pemulihan
- Mengelola aliran data antar fungsi

Pendekatan yang direkomendasikan

Gunakan dua Fungsi Acara Lambda: satu untuk memvalidasi pesanan dan satu untuk memproses pembayaran. Step Functions mengkoordinasikan fungsi-fungsi ini dengan:

- Menjalankan tugas dalam urutan yang benar
- Melewati data antar fungsi
- Menerapkan penanganan kesalahan pada setiap langkah
- Menggunakan status Pilihan untuk memastikan hanya pesanan yang valid yang dilanjutkan ke pembayaran

Example grafik alur kerja



> **Note**
>
> Code-first alternatif: Untuk pemrosesan berurutan dengan checkpointing berbasis kode dan coba lagi, lihat Lambda langkah fungsi tahan lama.

Penanganan kesalahan yang kompleks

Sementara Lambda menyediakan kemampuan coba ulang untuk pemanggilan asinkron dan pemetaan sumber peristiwa, Step Functions menawarkan penanganan kesalahan yang lebih canggih untuk alur kerja yang kompleks. Anda dapat mengonfigurasi percobaan ulang otomatis dengan backoff eksponensial dan menetapkan kebijakan coba ulang yang berbeda untuk berbagai jenis kesalahan. Saat percobaan ulang habis, gunakan `Catch` untuk merutekan kesalahan ke status

fallback. Ini sangat berguna ketika Anda memerlukan penanganan kesalahan tingkat alur kerja yang mengoordinasikan beberapa fungsi dan layanan.

Untuk mempelajari selengkapnya tentang penanganan kesalahan Fungsi Peristiwa Lambda di mesin status, lihat Menangani kesalahan di Lokakarya. AWS Step Functions

Anti-pattern contoh

Satu Fungsi Acara Lambda menangani semua hal berikut:

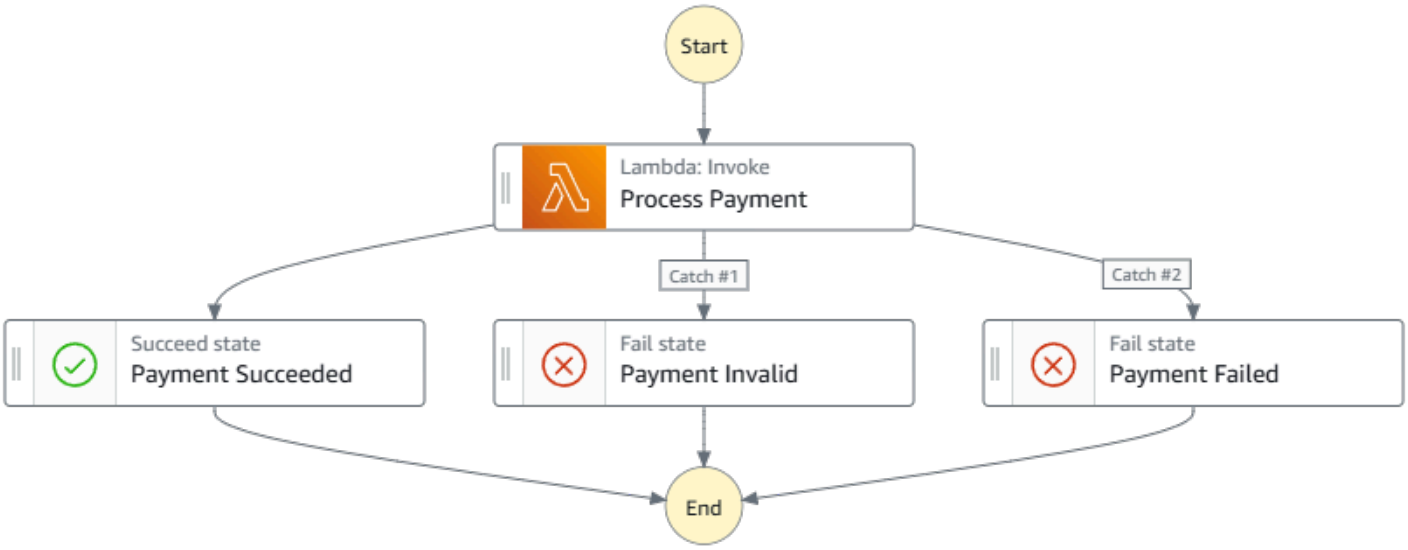
- Upaya untuk memanggil layanan pemrosesan pembayaran
- Jika layanan pembayaran tidak tersedia, fungsi menunggu dan mencoba lagi nanti.
- Menerapkan backoff eksponensial khusus untuk waktu tunggu
- Setelah semua upaya gagal, catch error dan pilih flow lain

Pendekatan yang direkomendasikan

Gunakan satu Fungsi Acara Lambda yang hanya berfokus pada pemrosesan pembayaran. Step Functions mengelola penanganan kesalahan dengan:

- Secara otomatis mencoba kembali tugas yang gagal dengan periode backoff yang dapat dikonfigurasi
- Menerapkan kebijakan coba ulang yang berbeda berdasarkan jenis kesalahan
- Merutekan berbagai jenis kesalahan ke status fallback yang sesuai
- Mempertahankan status dan riwayat penanganan kesalahan

Example grafik alur kerja



> **Note**
>
> Code-first alternatif: Fungsi tahan lama menyediakan penanganan kesalahan try-catch dengan strategi coba ulang yang dapat dikonfigurasi. Lihat Penanganan kesalahan dalam fungsi tahan lama.

Alur kerja bersyarat dan persetujuan manusia

Gunakan status Step Functions Choice untuk merutekan alur kerja berdasarkan output fungsi dan ForTaskToken akhiran tunggu untuk menjeda alur kerja untuk keputusan manusia. Misalnya, untuk memproses permintaan kenaikan batas kredit, gunakan Fungsi Peristiwa Lambda untuk mengevaluasi faktor risiko. Kemudian, gunakan Step Functions untuk merutekan permintaan berisiko tinggi ke persetujuan manual dan permintaan berisiko rendah ke persetujuan otomatis.

Untuk menerapkan contoh alur kerja yang menggunakan pola integrasi token tugas callback, lihat Callback with Task Token di Workshop. AWS Step Functions

Anti-pattern contoh

Satu Fungsi Peristiwa Lambda mengelola alur kerja persetujuan yang kompleks dengan:

- Menerapkan logika bersyarat bersarang untuk mengevaluasi permintaan kredit
- Memanggil fungsi persetujuan yang berbeda berdasarkan jumlah permintaan

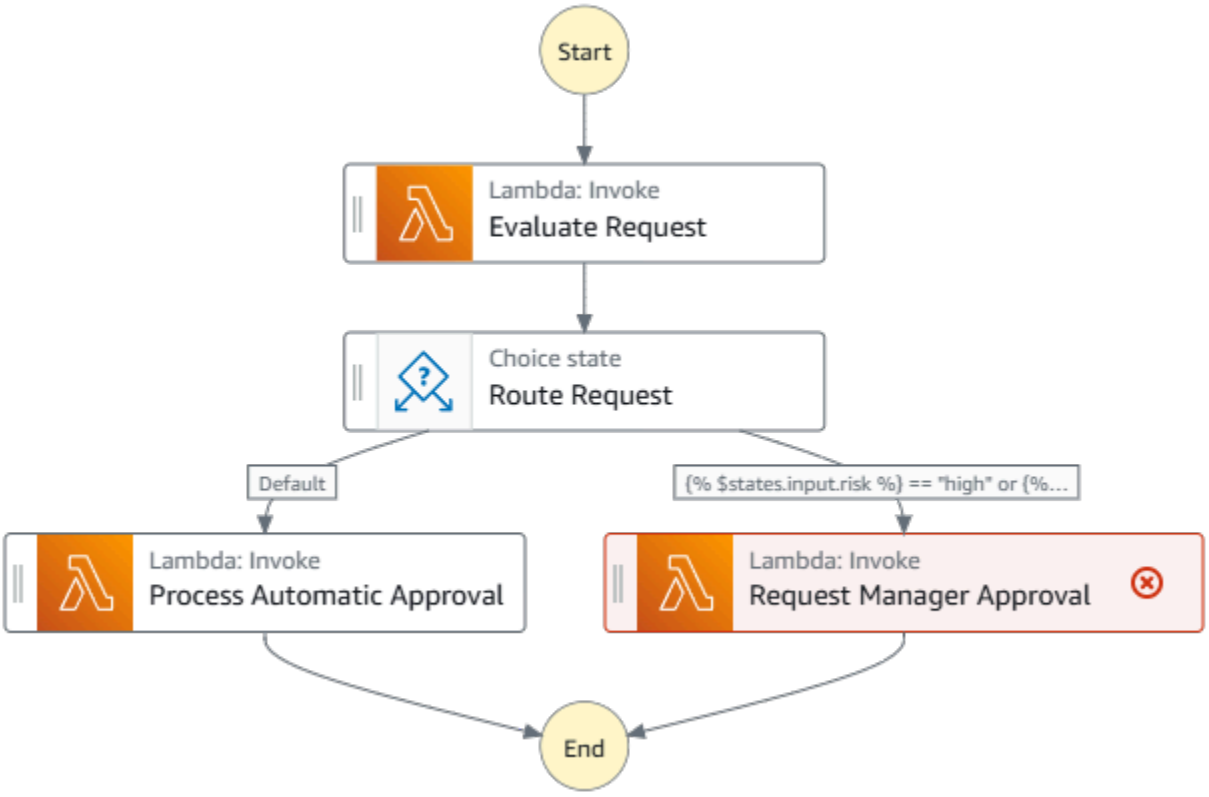
- Mengelola beberapa jalur persetujuan dan poin keputusan
- Melacak status persetujuan yang tertunda
- Menerapkan batas waktu dan logika notifikasi untuk persetujuan

Pendekatan yang direkomendasikan

Gunakan tiga Fungsi Acara Lambda: satu untuk mengevaluasi risiko setiap permintaan, satu untuk menyetujui permintaan berisiko rendah, dan satu untuk mengarahkan permintaan berisiko tinggi ke manajer untuk ditinjau. Step Functions mengelola alur kerja dengan:

- Menggunakan status Pilihan untuk merutekan permintaan berdasarkan jumlah dan tingkat risiko
- Menjeda eksekusi sambil menunggu persetujuan manusia
- Mengelola batas waktu untuk persetujuan yang tertunda
- Memberikan visibilitas ke status saat ini dari setiap permintaan

Example grafik alur kerja



> **Note**
>
> Code-first alternatif: Fungsi tahan lama mendukung callback untuk alur kerja manusia-in-the-loop. Lihat Callback dalam fungsi tahan lama.

Pemrosesan paralel

Step Functions menyediakan tiga cara untuk menangani pemrosesan paralel:

- Status Paralel mengeksekusi beberapa cabang alur kerja Anda secara bersamaan. Gunakan ini saat Anda perlu menjalankan fungsi yang berbeda secara paralel, seperti membuat thumbnail saat mengekstrak metadata gambar.
- Status Peta Inline memproses array data dengan hingga 40 iterasi bersamaan. Gunakan ini untuk kumpulan data kecil hingga menengah di mana Anda perlu melakukan operasi yang sama pada setiap item.
- Status Peta Terdistribusi menangani pemrosesan paralel skala besar dengan hingga 10.000 eksekusi bersamaan, mendukung array JSON dan sumber data Amazon Simple Storage Service (Amazon S3). Gunakan ini saat memproses kumpulan data besar atau saat Anda membutuhkan konkurensi yang lebih tinggi.

Anti-pattern contoh

Satu Fungsi Peristiwa Lambda mencoba mengelola pemrosesan paralel dengan:

- Secara bersamaan menjalankan beberapa fungsi pemrosesan gambar
- Menerapkan logika eksekusi parallel kustom
- Mengelola batas waktu dan penanganan kesalahan untuk setiap tugas paralel
- Mengumpulkan dan menggabungkan hasil dari semua fungsi

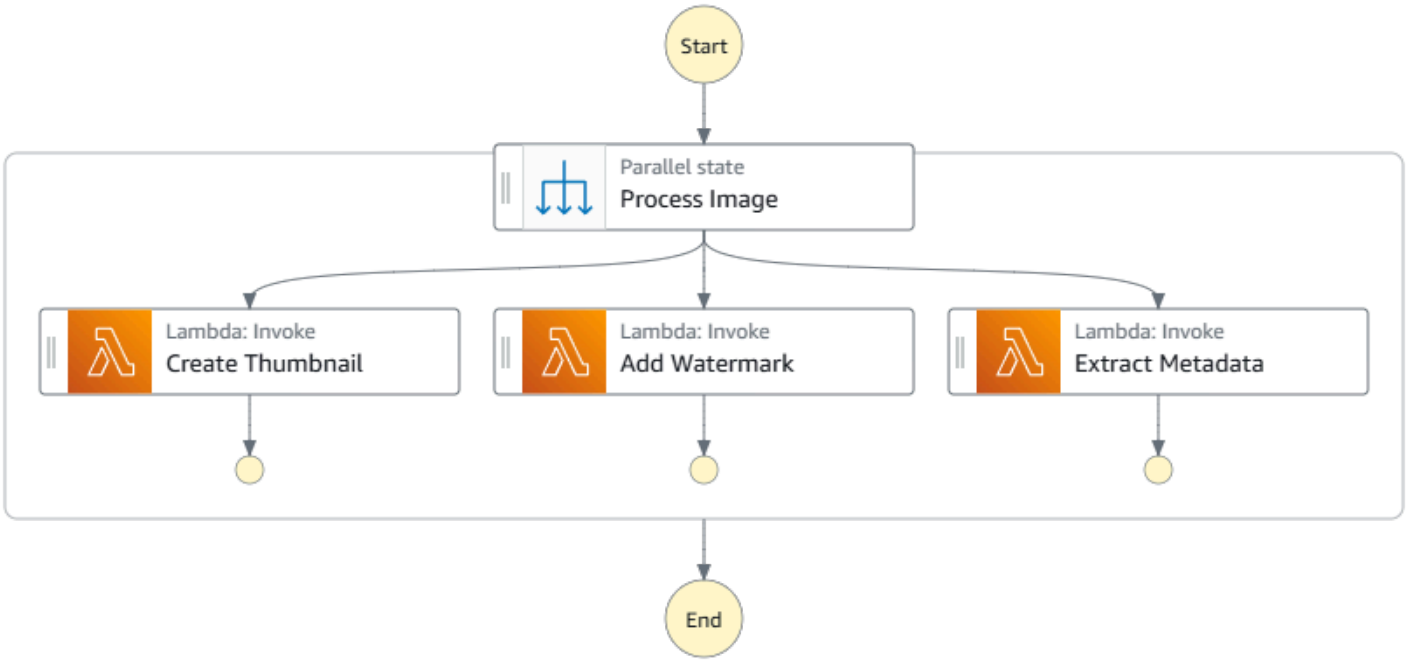
Pendekatan yang direkomendasikan

Gunakan tiga Fungsi Acara Lambda: satu untuk membuat gambar mini, satu untuk menambahkan tanda air, dan satu untuk mengekstrak metadata. Step Functions mengelola fungsi-fungsi ini dengan:

- Menjalankan semua fungsi secara bersamaan menggunakan status Paralel
- Mengumpulkan hasil dari setiap fungsi ke dalam array yang diurutkan

- Mengelola batas waktu dan penanganan kesalahan di semua eksekusi paralel
- Melanjutkan hanya ketika semua cabang paralel selesai

Example grafik alur kerja



> **Note**
>
> Code-first alternatif: Fungsi `parallel()` dan `map()` pengoperasian yang tahan lama. Lihat Eksekusi paralel.

Kapan tidak menggunakan Step Functions dengan Lambda

Tidak semua Lambda-based aplikasi mendapat manfaat dari penggunaan Step Functions. Pertimbangkan skenario ini saat memilih arsitektur aplikasi Anda.

- Aplikasi sederhana
- Pemrosesan data yang kompleks
- CPU-intensive beban kerja

Aplikasi sederhana

> **Note**
>
> Untuk alur kerja yang tidak memerlukan desain visual atau integrasi layanan ekstensif, fungsi Lambda yang tahan lama mungkin merupakan alternatif yang lebih sederhana yang menjaga logika alur kerja dalam kode di dalam Lambda.

Untuk aplikasi yang tidak memerlukan orkestrasi kompleks, menggunakan Step Functions mungkin menambah kompleksitas yang tidak perlu. Misalnya, jika Anda hanya memproses pesan dari antrean Amazon SQS atau menanggapi peristiwa Amazon EventBridge , Anda dapat mengonfigurasi layanan ini untuk memanggil Fungsi Peristiwa Lambda Anda secara langsung. Demikian pula, jika aplikasi Anda hanya terdiri dari satu atau dua Fungsi Peristiwa Lambda dengan penanganan kesalahan langsung, pemanggilan Lambda langsung atau arsitektur berbasis peristiwa mungkin lebih mudah untuk diterapkan dan dipelihara.

Pemrosesan data yang kompleks

Anda dapat menggunakan status Peta Terdistribusi Step Functions untuk memproses kumpulan data Amazon S3 besar secara bersamaan dengan Lambda Event Functions. Ini efektif untuk banyak beban kerja paralel skala besar, termasuk memproses data semi-terstruktur seperti file JSON atau CSV. Namun, untuk transformasi data yang lebih kompleks atau analitik lanjutan, pertimbangkan alternatif ini:

- Pipa transformasi data: Digunakan AWS Glue untuk pekerjaan ETL yang memproses data terstruktur atau semi-terstruktur dari berbagai sumber. AWS Glue sangat berguna ketika Anda membutuhkan katalog data bawaan dan kemampuan manajemen skema.
- Analisis data: Gunakan EMR Amazon untuk analitik data skala petabyte, terutama saat Anda membutuhkan alat Apache Hadoop ekosistem atau untuk beban kerja pembelajaran mesin yang melebihi batas memori Lambda.

CPU-intensive beban kerja

Meskipun Step Functions dapat mengatur tugas CPU-intensive , Lambda Event Functions mungkin tidak cocok untuk beban kerja ini karena sumber daya CPU-nya yang terbatas. Untuk operasi komputasi intensif dalam alur kerja Anda, pertimbangkan alternatif ini:

- Orkestrasi kontainer: Gunakan Step Functions untuk mengelola tugas Amazon Elastic Container Service (Amazon ECS) untuk sumber daya komputasi yang lebih konsisten dan terukur.
- Pemrosesan Batch: Integrasikan AWS Batch dengan Step Functions untuk mengelola pekerjaan batch intensif komputasi yang memerlukan penggunaan CPU berkelanjutan.

Memanggil Fungsi Acara Lambda dengan acara batch Amazon S3

Anda dapat menggunakan operasi batch Amazon S3 untuk menjalankan Fungsi Peristiwa Lambda pada sekumpulan besar objek Amazon S3. Amazon S3 melacak kemajuan operasi batch, mengirimkan pemberitahuan, dan menyimpan laporan penyelesaian yang menunjukkan status setiap tindakan.

Untuk menjalankan operasi batch, Anda membuat pekerjaan operasi batch Amazon S3. Saat Anda membuat pekerjaan, Anda memberikan manifest (daftar objek) dan mengonfigurasi tindakan yang akan dilakukan pada objek tersebut.

Saat pekerjaan batch dimulai, Amazon S3 memanggil Fungsi Peristiwa Lambda secara sinkron untuk setiap objek dalam manifes. Parameter kejadian mencakup nama bucket dan objek.

Contoh berikut menunjukkan peristiwa yang dikirimkan Amazon S3 ke Fungsi Peristiwa Lambda untuk objek yang diberi nama pelanggan Image1.jpg di bucket amzn-s3-demo-bucket.

Example Kejadian permintaan batch Amazon S3

```
{
"invocationSchemaVersion": "1.0",
    "invocationId": "YXNkbGZqYWRmaiBhc2RmdW9hZHNmZGpmaGFzbGtkaGZza2RmaAo",
    "job": {
        "id": "f3cc4f60-61f6-4a2b-8a21-d07600c373ce"
    },
    "tasks": [
        {
            "taskId": "dGFza2lkZ29lc2hlcmUK",
            "s3Key": "customerImage1.jpg",
            "s3VersionId": "1",
            "s3BucketArn": "arn:aws:s3:::amzn-s3-demo-bucket"
        }
    ]
}
```

Fungsi Peristiwa Lambda Anda harus mengembalikan objek JSON dengan bidang seperti yang ditunjukkan pada contoh berikut. Anda dapat menyalin `invocationId` dan `taskId` dari parameter kejadian. Anda dapat mengembalikan string di `resultString`. Amazon S3 menyimpan nilai `resultString` dalam laporan penyelesaian.

Example Respons permintaan batch Amazon S3

```
{
  "invocationSchemaVersion": "1.0",
  "treatMissingKeysAs" : "PermanentFailure",
  "invocationId" : "YXNkbGZqYWRmaiBhc2RmdW9hZHNmZGpmaGFzbGtkaGZza2RmaAo",
  "results": [
    {
      "taskId": "dGFza2lkZ29lc2hlcmUK",
      "resultCode": "Succeeded",
      "resultString": "[\"Alice\", \"Bob\"]"
    }
  ]
}
```

Memanggil Fungsi Acara Lambda dari operasi batch Amazon S3

Anda dapat memanggil Fungsi Acara Lambda dengan ARN fungsi yang tidak memenuhi syarat atau memenuhi syarat. Jika Anda ingin menggunakan versi fungsi yang sama untuk seluruh pekerjaan batch, konfigurasikan versi fungsi tertentu di parameter `FunctionARN` saat membuat pekerjaan Anda. Jika Anda mengonfigurasi sebuah alias atau pengualifikasi $LATEST, pekerjaan batch segera mulai memanggil versi baru dari fungsi tersebut jika alias atau $LATEST diperbarui selama pelaksanaan pekerjaan.

Perhatikan bahwa Anda tidak dapat menggunakan kembali fungsi berbasis kejadian Amazon S3 yang sudah ada untuk operasi batch. Ini karena operasi batch Amazon S3 meneruskan parameter peristiwa yang berbeda ke Fungsi Peristiwa Lambda dan mengharapkan pesan balasan dengan struktur JSON tertentu.

Dalam kebijakan berbasis sumber daya yang Anda buat untuk pekerjaan batch Amazon S3, pastikan Anda menetapkan izin untuk pekerjaan tersebut untuk menjalankan Fungsi Peristiwa Lambda.

Di peran eksekusi untuk fungsi tersebut, tetapkan kebijakan kepercayaan bagi Amazon S3 untuk mengambil peran tersebut saat menjalankan fungsi Anda.

Jika fungsi Anda menggunakan AWS SDK untuk mengelola resource Amazon S3, Anda perlu menambahkan izin Amazon S3 dalam peran eksekusi.

Saat pekerjaan berjalan, Amazon S3 memulai beberapa instans fungsi untuk memproses objek Amazon S3 secara paralel, hingga mencapai batas konkurensi fungsi. Amazon S3 membatasi peningkatan instans awal guna menghindari biaya berlebih untuk pekerjaan kecil.

Jika Fungsi Peristiwa Lambda mengembalikan kode `TemporaryFailure` respons, Amazon S3 akan mencoba ulang operasinya.

Untuk informasi lebih lanjut tentang operasi batch Amazon S3, lihat Melakukan operasi batch dalam Panduan Developer Amazon S3.

Untuk contoh cara menggunakan Fungsi Peristiwa Lambda di operasi batch Amazon S3, lihat Memanggil fungsi Lambda dari operasi batch Amazon S3 di Panduan Pengembang Amazon S3.

Memanggil Fungsi Acara Lambda dengan notifikasi Amazon SNS

Anda dapat menggunakan Fungsi Peristiwa Lambda untuk memproses notifikasi Amazon Simple Notification Service (Amazon SNS). Amazon SNS mendukung Lambda Event Functions sebagai target untuk pesan yang dikirim ke topik. Anda dapat melanggankan fungsi Anda ke topik di akun yang sama atau di akun AWS lainnya. Untuk panduan terperinci, lihat. the section called “Tutorial”

Lambda mendukung pemicu SNS untuk topik SNS standar saja. Topik FIFO tidak didukung.

Lambda memproses pesan SNS secara asinkron dengan mengantri pesan dan menangani percobaan ulang. Jika Amazon SNS tidak dapat mencapai Lambda atau pesan ditolak, Amazon SNS melakukan percobaan ulang dengan interval yang diperpanjang selama beberapa jam. Untuk perinciannya, lihat Reliabilitas di FAQ Amazon SNS.

> **Warning**
>
> Pemanggilan asinkron Lambda memproses setiap peristiwa setidaknya sekali, dan pemrosesan duplikat catatan dapat terjadi. Untuk menghindari potensi masalah yang terkait dengan duplikat peristiwa, kami sangat menyarankan agar Anda membuat kode fungsi Anda idempoten. Untuk mempelajari lebih lanjut, lihat Bagaimana cara membuat fungsi Lambda saya idempoten di Pusat Pengetahuan. AWS

Utilitas idempotensi dari Powertools untuk AWS Lambda

Utilitas idempotensi dari Powertools untuk Lambda membuat Fungsi Acara AWS Lambda Anda idempoten. Ini tersedia untuk Python,, Java TypeScript, dan .NET. Untuk informasi lebih lanjut, lihat utilitas Idempotency dalam dokumentasi Powertools for Lambda AWS (Python), Idempotency Utility dalam dokumentasi Powertools for Lambda (), Idempotency Utility dalam dokumentasi Powertools for AWS Lambda (JavaTypeScript), dan Idempotency Utility dalam dokumentasi Powertools for AWS Lambda (.NET). AWS

Topik

- Menambahkan pemicu topik Amazon SNS untuk Fungsi Peristiwa Lambda menggunakan konsol
- Menambahkan pemicu topik Amazon SNS secara manual untuk Fungsi Peristiwa Lambda
- Contoh bentuk acara SNS
- Tutorial: Menggunakan AWS Lambda dengan Layanan Pemberitahuan Sederhana Amazon

Menambahkan pemicu topik Amazon SNS untuk Fungsi Peristiwa Lambda menggunakan konsol

Untuk menambahkan topik SNS sebagai pemicu Fungsi Peristiwa Lambda, cara termudah adalah dengan menggunakan konsol Lambda. Saat Anda menambahkan pemicu melalui konsol, Lambda secara otomatis mengatur izin dan langganan yang diperlukan untuk mulai menerima acara dari topik SNS.

Untuk menambahkan topik SNS sebagai pemicu Fungsi Peristiwa Lambda (konsol)

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih nama fungsi yang ingin Anda tambahkan pemicunya.
3. Pilih Konfigurasi, lalu pilih Pemicu.
4. Pilih Tambahkan pemicu.
5. Di bawah konfigurasi Trigger, di menu tarik-turun, pilih SNS.
6. Untuk topik SNS, pilih topik SNS untuk berlangganan.

Menambahkan pemicu topik Amazon SNS secara manual untuk Fungsi Peristiwa Lambda

Untuk mengatur pemicu SNS untuk Fungsi Peristiwa Lambda secara manual, Anda harus menyelesaikan langkah-langkah berikut:

- Tentukan kebijakan berbasis sumber daya untuk fungsi Anda agar SNS dapat memanggilnya.
- Berlangganan Fungsi Acara Lambda Anda ke topik Amazon SNS.

> **Note**
>
> Jika topik SNS dan Fungsi Acara Lambda Anda berada di akun AWS yang berbeda, Anda juga perlu memberikan izin tambahan untuk mengizinkan langganan lintas akun ke topik SNS. Untuk informasi selengkapnya, lihat Memberikan izin lintas akun untuk langganan Amazon SNS.

Anda dapat menggunakan AWS Command Line Interface (AWS CLI) untuk menyelesaikan kedua langkah ini. Pertama, untuk menentukan kebijakan berbasis sumber daya untuk Fungsi Peristiwa

Lambda yang memungkinkan pemanggilan SNS, gunakan perintah berikut. AWS CLI Pastikan untuk mengganti nilai `--function-name` dengan nama Lambda Event Function Anda, dan nilai `--source-arn` dengan topik SNS ARN Anda.

```
aws lambda add-permission --function-name example-function \
    --source-arn arn:aws:sns:us-east-1:123456789012:sns-topic-for-lambda \
    --statement-id function-with-sns --action "lambda:InvokeFunction" \
    --principal sns.amazonaws.com
```

Untuk berlangganan fungsi Anda ke topik SNS, gunakan AWS CLI perintah berikut. Ganti nilai `--topic-arn` dengan ARN topik SNS Anda, dan nilainya dengan ARN Fungsi `--notification-endpoint` Acara Lambda Anda.

```
aws sns subscribe --protocol lambda \
    --region us-east-1 \
    --topic-arn arn:aws:sns:us-east-1:123456789012:sns-topic-for-lambda \
    --notification-endpoint arn:aws:lambda:us-east-1:123456789012:function:example-
function
```

Contoh bentuk acara SNS

Amazon SNS memanggil fungsi Anda secara asinkron dengan kejadian yang berisi pesan dan metadata.

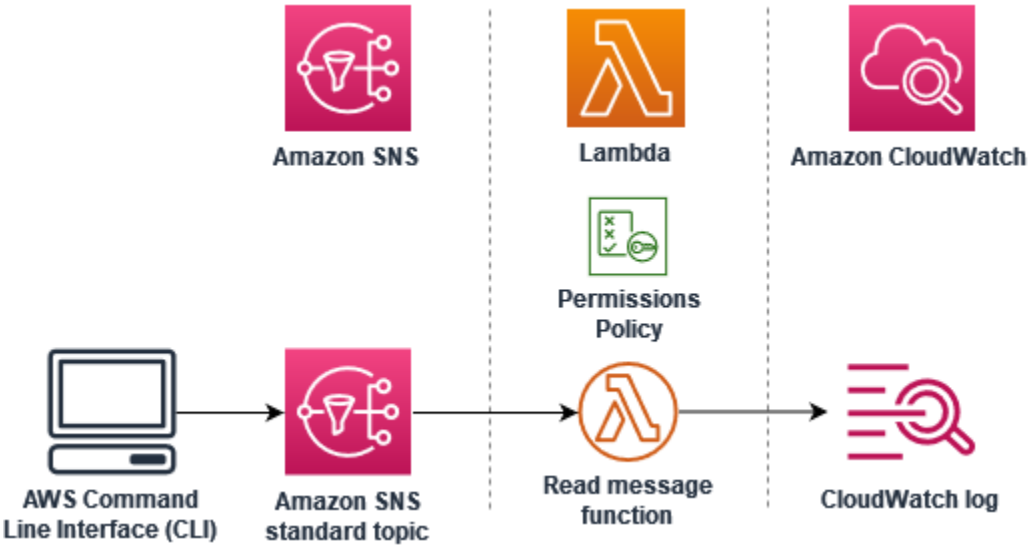
Example Kejadian pesan Amazon SNS

```
{
  "Records": [
    {
      "EventVersion": "1.0",
      "EventSubscriptionArn": "arn:aws:sns:us-east-1:123456789012:sns-lambda:21be56ed-
a058-49f5-8c98-aedd2564c486",
      "EventSource": "aws:sns",
      "Sns": {
        "SignatureVersion": "1",
        "Timestamp": "2019-01-02T12:45:07.000Z",
        "Signature": "tcc6faL2yUC6dgZdmrwh1Y4cGa/ebXEkAi6RibDsvpi+tE/1+82j...65r==",
        "SigningCertURL": "https://sns.us-east-1.amazonaws.com/
SimpleNotificationService-ac565b8b1a6c5d002d285f9598aa1d9b.pem",
        "MessageId": "95df01b4-ee98-5cb9-9903-4c221d41eb5e",
        "Message": "Hello from SNS!",
        "MessageAttributes": {
```

```
          "Test": {
            "Type": "String",
            "Value": "TestString"
          },
          "TestBinary": {
            "Type": "Binary",
            "Value": "TestBinary"
          }
        },
        "Type": "Notification",
        "UnsubscribeUrl": "https://sns.us-east-1.amazonaws.com/?
Action=Unsubscribe&amp;SubscriptionArn=arn:aws:sns:us-east-1:123456789012:test-
lambda:21be56ed-a058-49f5-8c98-aedd2564c486",
        "TopicArn":"arn:aws:sns:us-east-1:123456789012:sns-lambda",
        "Subject": "TestInvoke"
      }
    }
  ]
}
```

Tutorial: Menggunakan AWS Lambda dengan Layanan Pemberitahuan Sederhana Amazon

Dalam tutorial ini, Anda menggunakan Fungsi Acara Lambda dalam satu Akun AWS untuk berlangganan topik Amazon Simple Notification Service (Amazon SNS) secara terpisah. Akun AWS Saat memublikasikan pesan ke topik Amazon SNS, Fungsi Peristiwa Lambda akan membaca konten pesan dan mengeluarkannya ke Amazon Logs. CloudWatch Untuk menyelesaikan tutorial ini, Anda menggunakan AWS Command Line Interface (AWS CLI).



Untuk menyelesaikan tutorial ini, Anda melakukan langkah-langkah berikut:

- Di akun A, buat topik Amazon SNS.
- Di akun B, buat Fungsi Peristiwa Lambda yang membaca pesan dari topik.
- Di akun B, buat langganan ke topik.
- Publikasikan pesan ke topik Amazon SNS di akun A dan konfirmasikan bahwa Fungsi Peristiwa Lambda di akun B mengeluarkannya ke Log. CloudWatch

Dengan menyelesaikan langkah-langkah ini, Anda mempelajari cara mengonfigurasi topik Amazon SNS untuk menjalankan Fungsi Peristiwa Lambda. Anda juga mempelajari cara membuat kebijakan AWS Identity and Access Management (IAM) yang memberikan izin untuk sumber daya di sumber lain Akun AWS untuk memanggil Lambda.

Dalam tutorial, Anda menggunakan dua terpisah Akun AWS. AWS CLI Perintah mengilustrasikan ini dengan menggunakan dua profil bernama yang disebut `accountA` dan`accountB`, masing-masing dikonfigurasi untuk digunakan dengan yang berbeda Akun AWS. Untuk mempelajari cara mengonfigurasi penggunaan profil yang berbeda, lihat Pengaturan file konfigurasi dan kredensi di Panduan AWS Command Line Interface Pengguna untuk Versi 2. AWS CLI Pastikan untuk mengonfigurasi default yang sama Wilayah AWS untuk kedua profil.

Jika AWS CLI profil yang Anda buat untuk keduanya Akun AWS menggunakan nama yang berbeda, atau jika Anda menggunakan profil default dan satu profil bernama, ubah AWS CLI perintah dalam langkah-langkah berikut sesuai kebutuhan.

Prasyarat

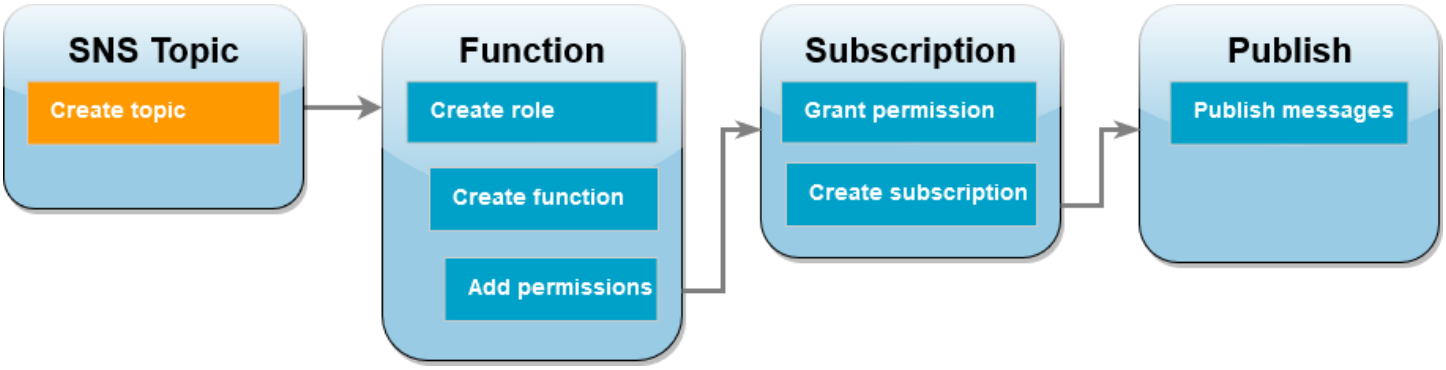
Instal AWS Command Line Interface

Jika Anda belum menginstal AWS Command Line Interface, ikuti langkah-langkah di Menginstal atau memperbarui versi terbaru AWS CLI untuk menginstalnya.

Tutorial ini membutuhkan terminal baris perintah atau shell untuk menjalankan perintah. Di Linux dan macOS, gunakan shell dan manajer paket pilihan Anda.

> **Note**
>
> Di Windows, beberapa perintah Bash CLI yang biasa Anda gunakan dengan Lambda (`zip`seperti) tidak didukung oleh terminal bawaan sistem operasi. Untuk mendapatkan Windows-integrated versi Ubuntu dan Bash, instal Subsistem Windows untuk Linux.

Buat topik Amazon SNS (akun A)



Untuk membuat topik

- Di akun A, buat topik standar Amazon SNS menggunakan perintah berikut AWS CLI .

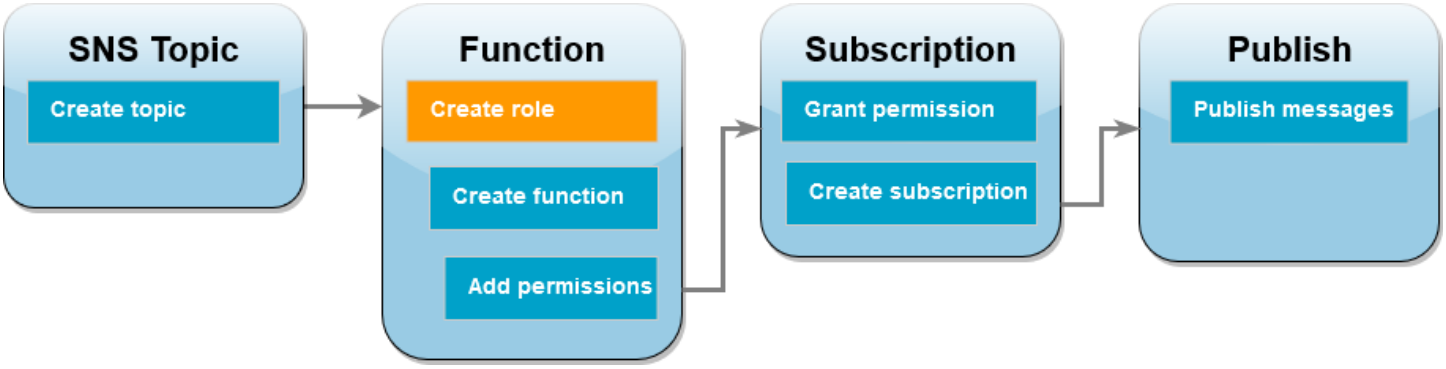
  ```
  aws sns create-topic --name sns-topic-for-lambda --profile accountA
  ```

  Anda akan melihat output seperti yang berikut ini.

  ```
  {
      "TopicArn": "arn:aws:sns:us-west-2:123456789012:sns-topic-for-lambda"
  }
  ```

  Catat Nama Sumber Daya Amazon (ARN) topik Anda. Anda akan membutuhkannya nanti dalam tutorial ketika Anda menambahkan izin ke Fungsi Acara Lambda Anda untuk berlangganan topik.

Buat peran eksekusi fungsi (akun B)

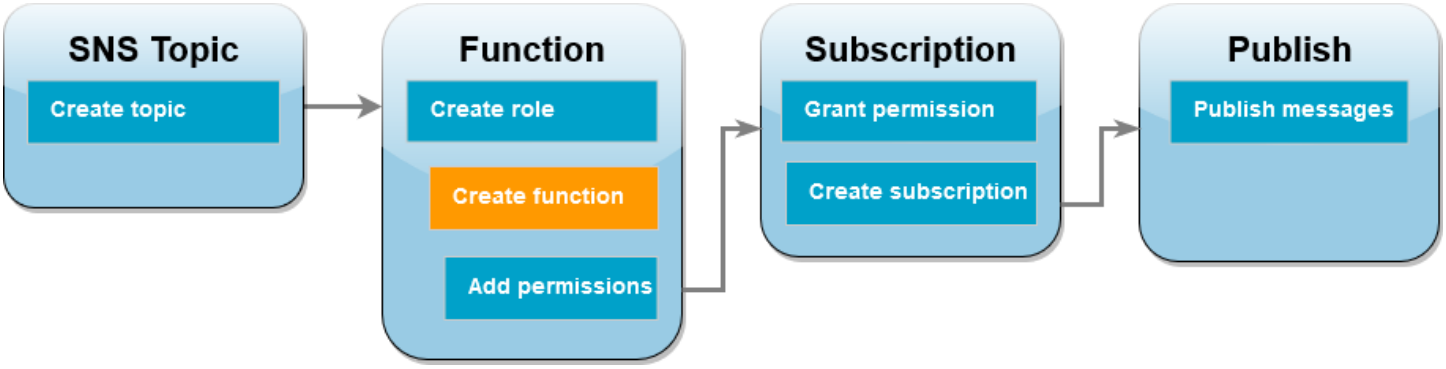


Peran eksekusi adalah peran IAM yang memberikan izin Fungsi Peristiwa Lambda untuk mengakses dan sumber daya. Layanan AWS Sebelum membuat fungsi di akun B, Anda membuat peran yang memberikan izin dasar fungsi untuk menulis CloudWatch log ke Log. Kami akan menambahkan izin untuk membaca dari topik Amazon SNS Anda di langkah selanjutnya.

Untuk membuat peran eksekusi

1. Di akun B buka halaman peran di konsol IAM.
2. Pilih Buat peran.
3. Untuk jenis entitas Tepercaya, pilih AWS layanan.
4. Untuk kasus penggunaan, pilih Lambda.
5. Pilih Berikutnya.
6. Tambahkan kebijakan izin dasar ke peran dengan melakukan hal berikut:
   a. Di kotak pencarian Kebijakan izin, masukkan**`AWSLambdaBasicExecutionRole`**.
   b. Pilih Berikutnya.
7. Selesaikan pembuatan peran dengan melakukan hal berikut:
   a. Di bawah Rincian peran, masukkan **`lambda-sns-role`** nama Peran.
   b. Pilih Buat peran.

Buat Fungsi Acara Lambda (akun B)



Buat Fungsi Peristiwa Lambda yang memproses pesan Amazon SNS Anda. Kode fungsi mencatat isi pesan dari setiap catatan ke Amazon CloudWatch Logs.

Tutorial ini menggunakan runtime Node.js 24, tetapi kami juga menyediakan kode contoh dalam bahasa runtime lainnya. Anda dapat memilih tab di kotak berikut untuk melihat kode runtime yang Anda minati. JavaScript Kode yang akan Anda gunakan dalam langkah ini adalah pada contoh pertama yang ditunjukkan di JavaScripttab.

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara SNS dengan Lambda menggunakan.NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using Amazon.Lambda.Core;
using Amazon.Lambda.SNSEvents;


// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri
```

```
namespace SnsIntegration;

public class Function
{
    public async Task FunctionHandler(SNSEvent evnt, ILambdaContext context)
    {
        foreach (var record in evnt.Records)
        {
            await ProcessRecordAsync(record, context);
        }
        context.Logger.LogInformation("done");
    }

    private async Task ProcessRecordAsync(SNSEvent.SNSRecord record,
 ILambdaContext context)
    {
        try
        {
            context.Logger.LogInformation($"Processed record
 {record.Sns.Message}");

            // TODO: Do interesting work based on the new message
            await Task.CompletedTask;
        }
        catch (Exception e)
        {
            //You can use Dead Letter Queue to handle failures. By configuring a
 Lambda DLQ.
            context.Logger.LogError($"An error occurred");
            throw;
        }
    }
}
```

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara SNS dengan Lambda menggunakan Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "fmt"

 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

func handler(ctx context.Context, snsEvent events.SNSEvent) {
 for _, record := range snsEvent.Records {
  processMessage(record)
 }
 fmt.Println("done")
}

func processMessage(record events.SNSEventRecord) {
 message := record.SNS.Message
 fmt.Printf("Processed message: %s\n", message)
 // TODO: Process your record here
}

func main() {
 lambda.Start(handler)
}
```

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara SNS dengan Lambda menggunakan Java.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.LambdaLogger;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.SNSEvent;
import com.amazonaws.services.lambda.runtime.events.SNSEvent.SNSRecord;


import java.util.Iterator;
import java.util.List;

public class SNSEventHandler implements RequestHandler<SNSEvent, Boolean> {
    LambdaLogger logger;

    @Override
    public Boolean handleRequest(SNSEvent event, Context context) {
        logger = context.getLogger();
        List<SNSRecord> records = event.getRecords();
        if (!records.isEmpty()) {
            Iterator<SNSRecord> recordsIter = records.iterator();
            while (recordsIter.hasNext()) {
                processRecord(recordsIter.next());
            }
        }
        return Boolean.TRUE;
    }

    public void processRecord(SNSRecord record) {
```

```
        try {
            String message = record.getSNS().getMessage();
            logger.log("message: " + message);
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }

}
```

JavaScript

SDK untuk JavaScript (v3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara SNS dengan JavaScript Lambda menggunakan.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const record of event.Records) {
    await processMessageAsync(record);
  }
  console.info("done");
};

async function processMessageAsync(record) {
  try {
    const message = JSON.stringify(record.Sns.Message);
    console.log(`Processed message ${message}`);
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
```

```
    throw err;
  }
}
```

Mengkonsumsi acara SNS dengan TypeScript Lambda menggunakan.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { SNSEvent, Context, SNSHandler, SNSEventRecord } from "aws-lambda";

export const functionHandler: SNSHandler = async (
  event: SNSEvent,
  context: Context
): Promise<void> => {
  for (const record of event.Records) {
    await processMessageAsync(record);
  }
  console.info("done");
};

async function processMessageAsync(record: SNSEventRecord): Promise<any> {
  try {
    const message: string = JSON.stringify(record.Sns.Message);
    console.log(`Processed message ${message}`);
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

PHP

SDK untuk PHP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara SNS dengan Lambda menggunakan PHP.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

/*
Since native PHP support for AWS Lambda is not available, we are utilizing Bref's
 PHP functions runtime for AWS Lambda.
For more information on Bref's PHP runtime for Lambda, refer to: https://bref.sh/
docs/runtimes/function

Another approach would be to create a custom runtime.
A practical example can be found here: https://aws.amazon.com/blogs/apn/aws-
lambda-custom-runtime-for-php-a-practical-example/
*/

// Additional composer packages may be required when using Bref or any other PHP
 functions runtime.
// require __DIR__ . '/vendor/autoload.php';

use Bref\Context\Context;
use Bref\Event\Sns\SnsEvent;
use Bref\Event\Sns\SnsHandler;

class Handler extends SnsHandler
{
    public function handleSns(SnsEvent $event, Context $context): void
    {
        foreach ($event->getRecords() as $record) {
            $message = $record->getMessage();

            // TODO: Implement your custom processing logic here
            // Any exception thrown will be logged and the invocation will be
 marked as failed

            echo "Processed Message: $message" . PHP_EOL;
        }
    }
}

return new Handler();
```

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara SNS dengan Lambda menggunakan Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def lambda_handler(event, context):
    for record in event['Records']:
        process_message(record)
    print("done")

def process_message(record):
    try:
        message = record['Sns']['Message']
        print(f"Processed message {message}")
        # TODO; Process your record here

    except Exception as e:
        print("An error occurred")
        raise e
```

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara SNS dengan Lambda menggunakan Ruby.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def lambda_handler(event:, context:)
  event['Records'].map { |record| process_message(record) }
end

def process_message(record)
  message = record['Sns']['Message']
  puts("Processing message: #{message}")
rescue StandardError => e
  puts("Error processing message: #{e}")
  raise
end
```

Rust

SDK for Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara SNS dengan Lambda menggunakan Rust.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::event::sns::SnsEvent;
use aws_lambda_events::sns::SnsRecord;
use lambda_runtime::{run, service_fn, Error, LambdaEvent};
use tracing::info;

// Built with the following dependencies:
//  aws_lambda_events = { version = "0.10.0", default-features = false, features
 = ["sns"] }
//  lambda_runtime = "0.8.1"
//  tokio = { version = "1", features = ["macros"] }
```

```
//  tracing = { version = "0.1", features = ["log"] }
//  tracing-subscriber = { version = "0.3", default-features = false, features =
 ["fmt"] }

async fn function_handler(event: LambdaEvent<SnsEvent>) -> Result<(), Error> {
    for event in event.payload.records {
        process_record(&event)?;
    }

    Ok(())
}

fn process_record(record: &SnsRecord) -> Result<(), Error> {
    info!("Processing SNS Message: {}", record.sns.message);

    // Implement your record handling code here.

    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        .with_target(false)
        .without_time()
        .init();

    run(service_fn(function_handler)).await
}
```

Untuk membuat Event Function

1. Buat direktori untuk proyek, dan kemudian beralih ke direktori itu.

   ```
   mkdir sns-tutorial
   cd sns-tutorial
   ```

2. Salin JavaScript kode sampel ke file baru bernama`index.js`.

3. Buat paket penyebaran menggunakan `zip` perintah berikut.

```
zip function.zip index.js
```

4. Jalankan AWS CLI perintah berikut untuk membuat Fungsi Acara Lambda Anda di akun B.

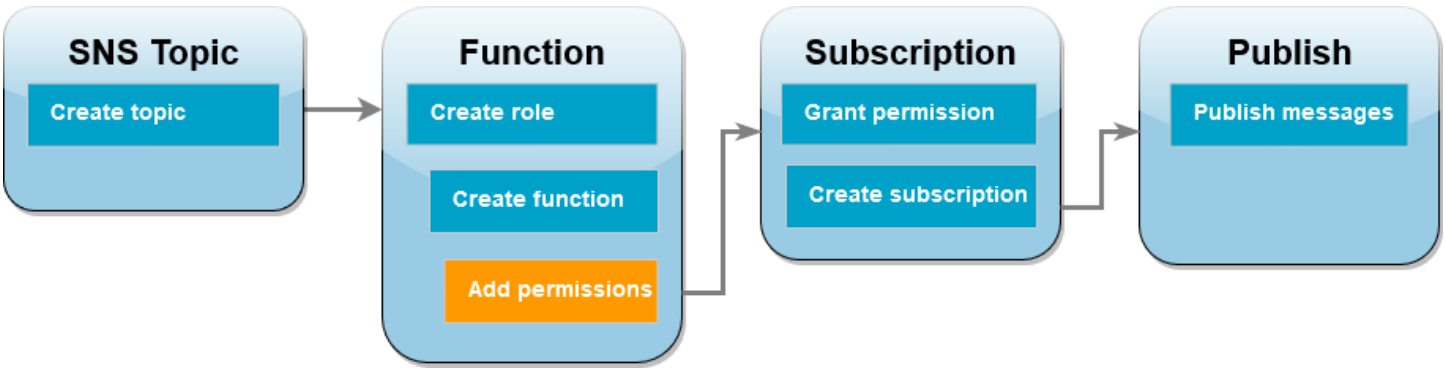
```
aws lambda create-function --function-name Function-With-SNS \
    --zip-file fileb://function.zip --handler index.handler --runtime nodejs24.x \
    --role arn:aws:iam::<AccountB_ID>:role/lambda-sns-role  \
    --timeout 60 --profile accountB
```

Anda akan melihat output seperti yang berikut ini.

```
{
    "FunctionName": "Function-With-SNS",
    "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:Function-With-
SNS",
    "Runtime": "nodejs24.x",
    "Role": "arn:aws:iam::123456789012:role/lambda_basic_role",
    "Handler": "index.handler",
    ...
    "RuntimeVersionConfig": {
        "RuntimeVersionArn": "arn:aws:lambda:us-
west-2::runtime:7d5f06b69c951da8a48b926ce280a9daf2e8bb1a74fc4a2672580c787d608206"
    }
}
```

5. Rekam Nama Sumber Daya Amazon (ARN) dari fungsi Anda. Anda akan membutuhkannya nanti di tutorial saat Anda menambahkan izin untuk mengizinkan Amazon SNS menjalankan fungsi Anda.

Tambahkan izin ke Fungsi Acara (akun B)



Agar Amazon SNS dapat menjalankan fungsi Anda, Anda harus memberinya izin dalam pernyataan tentang kebijakan berbasis sumber daya. Anda menambahkan pernyataan ini menggunakan AWS CLI `add-permission` perintah.

Untuk memberikan izin Amazon SNS untuk memanggil Fungsi Acara Anda

- Di akun B, jalankan AWS CLI perintah berikut menggunakan ARN untuk topik Amazon SNS yang Anda rekam sebelumnya.

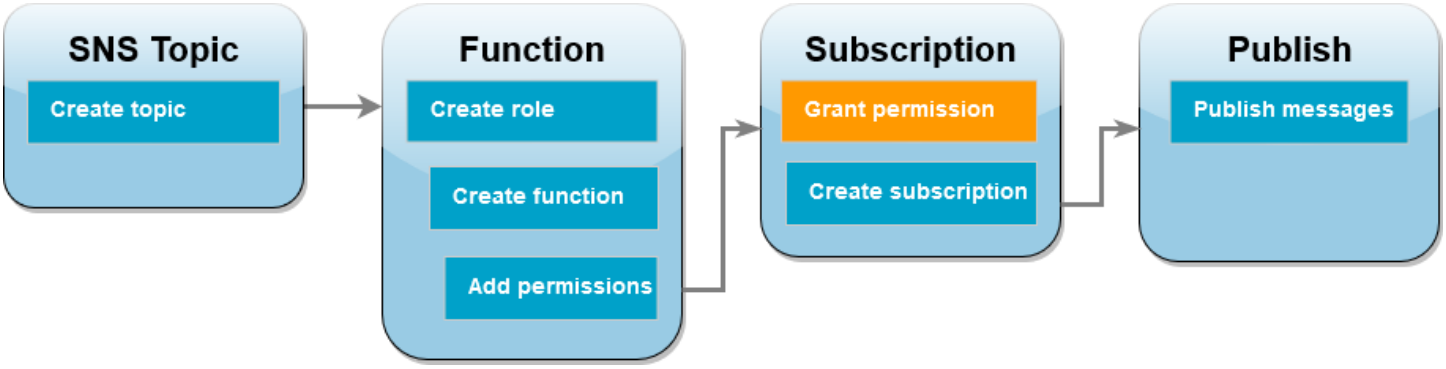
```
aws lambda add-permission --function-name Function-With-SNS \
    --source-arn arn:aws:sns:us-east-1:<AccountA_ID>:sns-topic-for-lambda \
    --statement-id function-with-sns --action "lambda:InvokeFunction" \
    --principal sns.amazonaws.com --profile accountB
```

  Anda akan melihat output seperti yang berikut ini.

```
{
    "Statement": "{\"Condition\":{\"ArnLike\":{\"AWS:SourceArn\":
      \"arn:aws:sns:us-east-1:<AccountA_ID>:sns-topic-for-lambda\"}},
      \"Action\":[\"lambda:InvokeFunction\"],
      \"Resource\":\"arn:aws:lambda:us-east-1:<AccountB_ID>:function:Function-With-
SNS\",
      \"Effect\":\"Allow\",\"Principal\":{\"Service\":\"sns.amazonaws.com\"},
      \"Sid\":\"function-with-sns\"}"
}
```

> **Note**
>
> Jika akun dengan topik Amazon SNS di-host dalam opt-in Wilayah AWS, Anda harus menentukan wilayah di prinsipal. Misalnya, jika Anda bekerja dengan topik Amazon SNS di wilayah Asia Pasifik (Hong Kong), Anda harus menentukan `sns.ap-east-1.amazonaws.com` bukan `sns.amazonaws.com` untuk prinsipal.

Berikan izin lintas akun untuk langganan Amazon SNS (akun A)



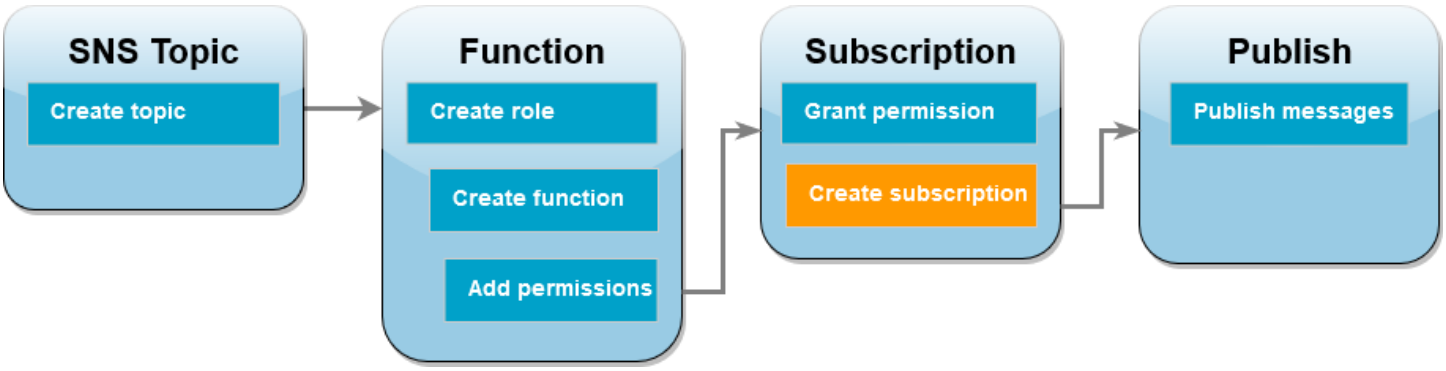
Agar Fungsi Acara Lambda Anda di akun B berlangganan topik Amazon SNS yang Anda buat di akun A, Anda harus memberikan izin kepada akun B untuk berlangganan topik Anda. Anda memberikan izin ini menggunakan AWS CLI `add-permission` perintah.

Untuk memberikan izin bagi akun B untuk berlangganan topik

- Di akun A, jalankan AWS CLI perintah berikut. Gunakan ARN untuk topik Amazon SNS yang Anda rekam sebelumnya.

```
aws sns add-permission --label lambda-access --aws-account-id <AccountB_ID> \
    --topic-arn arn:aws:sns:us-east-1:<AccountA_ID>:sns-topic-for-lambda \
    --action-name Subscribe ListSubscriptionsByTopic --profile accountA
```

Buat langganan (akun B)



Di akun B, Anda sekarang berlangganan Fungsi Acara Lambda Anda ke topik Amazon SNS yang Anda buat di awal tutorial di akun A. Saat pesan dikirim ke topik ini (**`sns-topic-for-lambda`**), Amazon SNS akan memanggil Fungsi Peristiwa Lambda Anda di akun B. **`Function-With-SNS`**

Untuk membuat langganan

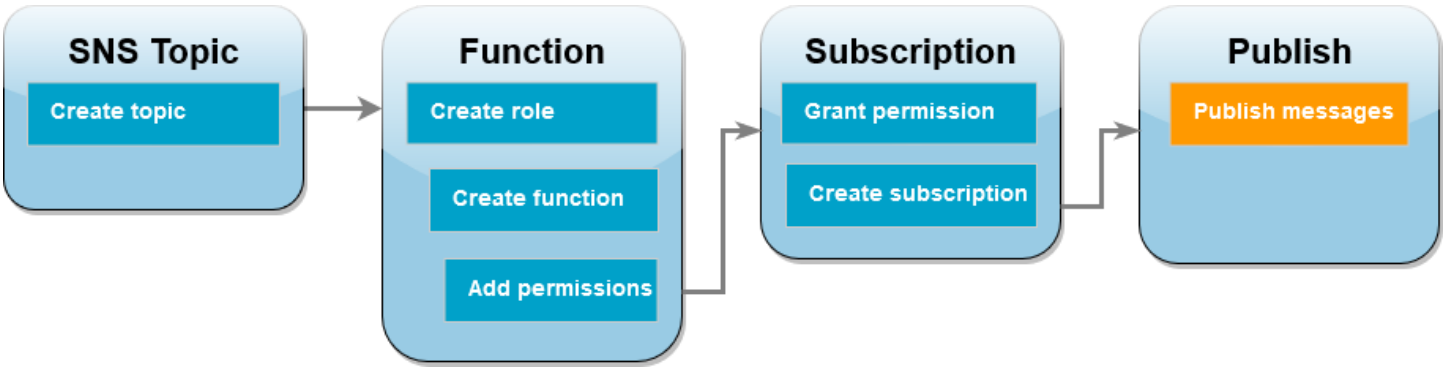
- Di akun B, jalankan AWS CLI perintah berikut. Gunakan wilayah default tempat Anda membuat topik dan ARN untuk topik dan Fungsi Acara Lambda.

```
aws sns subscribe --protocol lambda \
    --region us-east-1 \
    --topic-arn arn:aws:sns:us-east-1:<AccountA_ID>:sns-topic-for-lambda \
    --notification-endpoint arn:aws:lambda:us-
east-1:<AccountB_ID>:function:Function-With-SNS \
    --profile accountB
```

Anda akan melihat output seperti yang berikut ini.

```
{
    "SubscriptionArn": "arn:aws:sns:us-east-1:<AccountA_ID>:sns-topic-for-
lambda:5d906xxxx-7c8x-45dx-a9dx-0484e31c98xx"
}
```

Publikasikan pesan ke topik (akun A dan akun B)



Sekarang setelah Fungsi Acara Lambda Anda di akun B berlangganan topik Amazon SNS Anda di akun A, saatnya untuk menguji penyiapan Anda dengan menerbitkan pesan ke topik Anda. Untuk mengonfirmasi bahwa Amazon SNS telah memanggil Fungsi Peristiwa Lambda, Anda menggunakan CloudWatch Log untuk melihat output fungsi Anda.

Untuk mempublikasikan pesan ke topik Anda dan melihat output Fungsi Acara Anda

1. Masukkan `Hello World` ke dalam file teks dan simpan sebagai`message.txt`.

2. Dari direktori yang sama tempat Anda menyimpan file teks Anda, jalankan AWS CLI perintah berikut di akun A. Gunakan ARN untuk topik Anda sendiri.

```
aws sns publish --message file://message.txt --subject Test \
    --topic-arn arn:aws:sns:us-east-1:<AccountA_ID>:sns-topic-for-lambda \
    --profile accountA
```

   Ini mengembalikan ID pesan dengan pengenal unik, yang menunjukkan bahwa Amazon SNS telah menerima pesan tersebut. Amazon SNS kemudian mencoba mengirimkan pesan ke pelanggan topik. Untuk mengonfirmasi bahwa Amazon SNS telah memanggil Fungsi Peristiwa Lambda Anda, gunakan CloudWatch Log untuk melihat output fungsi Anda:

3. Di akun B, buka halaman Grup log di CloudWatch konsol Amazon.
4. Pilih grup log untuk fungsi Anda (`/aws/lambda/Function-With-SNS`).
5. Pilih aliran log terbaru.
6. Jika fungsi Anda dipanggil dengan benar, Anda akan melihat output yang mirip dengan berikut yang menunjukkan konten pesan yang Anda terbitkan ke topik Anda.

```
2023-07-31T21:42:51.250Z c1cba6b8-ade9-4380-aa32-d1a225da0e48 INFO Processed
 message Hello World
2023-07-31T21:42:51.250Z c1cba6b8-ade9-4380-aa32-d1a225da0e48 INFO done
```

Bersihkan sumber daya Anda

Sekarang Anda dapat menghapus sumber daya yang Anda buat untuk tutorial ini, kecuali Anda ingin mempertahankannya. Dengan menghapus AWS sumber daya yang tidak lagi Anda gunakan, Anda mencegah tagihan yang tidak perlu ke Anda Akun AWS.

Di Akun A, bersihkan topik Amazon SNS Anda.

Untuk menghapus topik Amazon SNS

1. Buka halaman Topik di konsol Amazon SNS.
2. Pilih topik yang Anda buat.
3. Pilih Hapus.
4. Masukkan **`delete me`** di bidang input teks.
5. Pilih Hapus.

Di Akun B, bersihkan peran eksekusi, Fungsi Acara Lambda, dan langganan Amazon SNS.

Untuk menghapus peran eksekusi

1. Buka halaman Peran dari konsol IAM.
2. Pilih peran eksekusi yang Anda buat.
3. Pilih Hapus.
4. Masukkan nama peran di bidang input teks dan pilih Hapus.

Untuk menghapus Fungsi Acara Lambda

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih Fungsi Acara yang Anda buat.
3. Pilih Tindakan, Hapus.
4. Ketik **`confirm`** kolom input teks dan pilih Hapus.

Untuk menghapus langganan Amazon SNS

1. Buka halaman Langganan di konsol Amazon SNS.
2. Pilih langganan yang Anda buat.
3. Pilih Delete, Delete.

Fungsi tahan lama Lambda

Dengan AWS Lambda fungsi tahan lama, Anda dapat membangun aplikasi multi-langkah tangguh dan alur kerja AI yang dapat dijalankan hingga satu tahun sambil mempertahankan kemajuan yang andal meskipun ada gangguan. Ketika fungsi tahan lama berjalan, siklus hidup lengkap ini disebut eksekusi tahan lama, yang menggunakan pos pemeriksaan untuk melacak kemajuan dan secara otomatis pulih dari kegagalan melalui pemutaran ulang, mengeksekusi ulang dari awal sambil melewatkan pekerjaan yang telah selesai.

Dalam setiap fungsi, Anda menggunakan operasi tahan lama sebagai blok bangunan mendasar. Langkah-langkah mengeksekusi logika bisnis dengan percobaan ulang bawaan dan pelacakan kemajuan, sementara menunggu menangguhkan eksekusi tanpa menimbulkan biaya komputasi, menjadikannya ideal untuk proses yang berjalan lama seperti alur kerja manusia-in-the-loop atau

polling dependensi eksternal. Baik Anda memproses pesanan, mengoordinasikan layanan mikro, atau mengatur aplikasi AI agentic, fungsi tahan lama mempertahankan status secara otomatis dan pulih dari kegagalan saat Anda menulis kode dalam bahasa pemrograman yang sudah dikenal.

> **Note**
>
> Panduan ini mencakup topik layanan Lambda seperti penerapan, konfigurasi, izin, pemanggilan, pemantauan, dan kuota. Untuk referensi pemrograman SDK, tutorial memulai cepat, dan panduan khusus bahasa, lihat AWS Panduan Pengembang SDK Eksekusi Tahan Lama.

Manfaat utama

Tulis kode tangguh secara alami: Dengan konstruksi pemrograman yang sudah dikenal, Anda menulis kode yang menangani kegagalan secara otomatis. Built-in checkpoint, percobaan ulang transparan, dan pemulihan otomatis berarti logika bisnis Anda tetap bersih dan fokus.

Bayar hanya untuk apa yang Anda gunakan: Selama operasi tunggu, fungsi Anda ditangguhkan tanpa menimbulkan biaya komputasi. Untuk alur kerja yang berjalan lama yang menunggu berjam-jam atau berhari-hari, Anda membayar hanya untuk waktu pemrosesan aktual, bukan menunggu siaga.

Kesederhanaan operasional: Dengan model tanpa server Lambda, Anda mendapatkan penskalaan otomatis, termasuk skala-ke-nol, tanpa mengelola infrastruktur. Fungsi tahan lama menangani manajemen status, logika coba ulang, dan pemulihan kegagalan secara otomatis, mengurangi overhead operasional.

Kapan menggunakan fungsi tahan lama

Short-lived koordinasi: Koordinasikan pembayaran, inventaris, dan pengiriman di berbagai layanan dengan rollback otomatis pada kegagalan. Memproses pesanan melalui validasi, otorisasi pembayaran, alokasi inventaris, dan pemenuhan dengan jaminan penyelesaian.

Memproses pembayaran dengan percaya diri: Bangun aliran pembayaran tangguh yang mempertahankan status transaksi melalui kegagalan dan menangani percobaan ulang secara otomatis. Koordinasikan otorisasi multi-langkah, pemeriksaan penipuan, dan penyelesaian di seluruh penyedia pembayaran dengan auditabilitas penuh di seluruh langkah.

Bangun alur kerja AI yang andal: Buat alur kerja AI multi-langkah yang merangkai panggilan model, menggabungkan umpan balik manusia, dan menangani tugas yang berjalan lama secara deterministik selama kegagalan. Secara otomatis melanjutkan setelah penangguhan, dan hanya membayar untuk waktu eksekusi aktif.

Mengatur pemenuhan pesanan yang kompleks: Koordinasikan pemrosesan pesanan di seluruh sistem inventaris, pembayaran, pengiriman, dan notifikasi dengan ketahanan bawaan. Secara otomatis menangani kegagalan sebagian, mempertahankan status pesanan meskipun ada gangguan, dan secara efisien menunggu peristiwa eksternal tanpa menghabiskan sumber daya komputasi.

Mengotomatiskan alur kerja bisnis multi-langkah: Bangun alur kerja yang andal untuk orientasi karyawan, persetujuan pinjaman, dan proses kepatuhan yang berlangsung berhari-hari atau berminggu-minggu. Pertahankan status alur kerja di seluruh persetujuan manusia, integrasi sistem, dan tugas terjadwal sambil memberikan visibilitas penuh ke dalam status dan riwayat proses.

Bagaimana fungsi tahan lama dibandingkan dengan Fungsi Langkah

Fungsi tahan lama dan Fungsi Langkah menyediakan orkestrasi alur kerja dengan manajemen status otomatis. Perbedaan utama adalah di mana mereka berjalan dan bagaimana Anda mendefinisikan alur kerja:

- Fungsi tahan lama: Jalankan dalam Lambda, gunakan bahasa pemrograman standar, dikelola dalam lingkungan Lambda
- Fungsi Langkah: Layanan mandiri, DSL berbasis grafik atau perancang visual, dikelola sepenuhnya tanpa pemeliharaan

Fungsi tahan lama sangat ideal untuk pengembangan aplikasi di Lambda di mana alur kerja digabungkan erat dengan logika bisnis. Step Functions unggul dalam orkestrasi alur kerja di seluruh AWS layanan di mana Anda membutuhkan desain visual, integrasi asli ke 220+ layanan, dan infrastruktur tanpa perawatan.

Untuk perbandingan terperinci, lihat Fungsi tahan lama atau Fungsi Langkah.

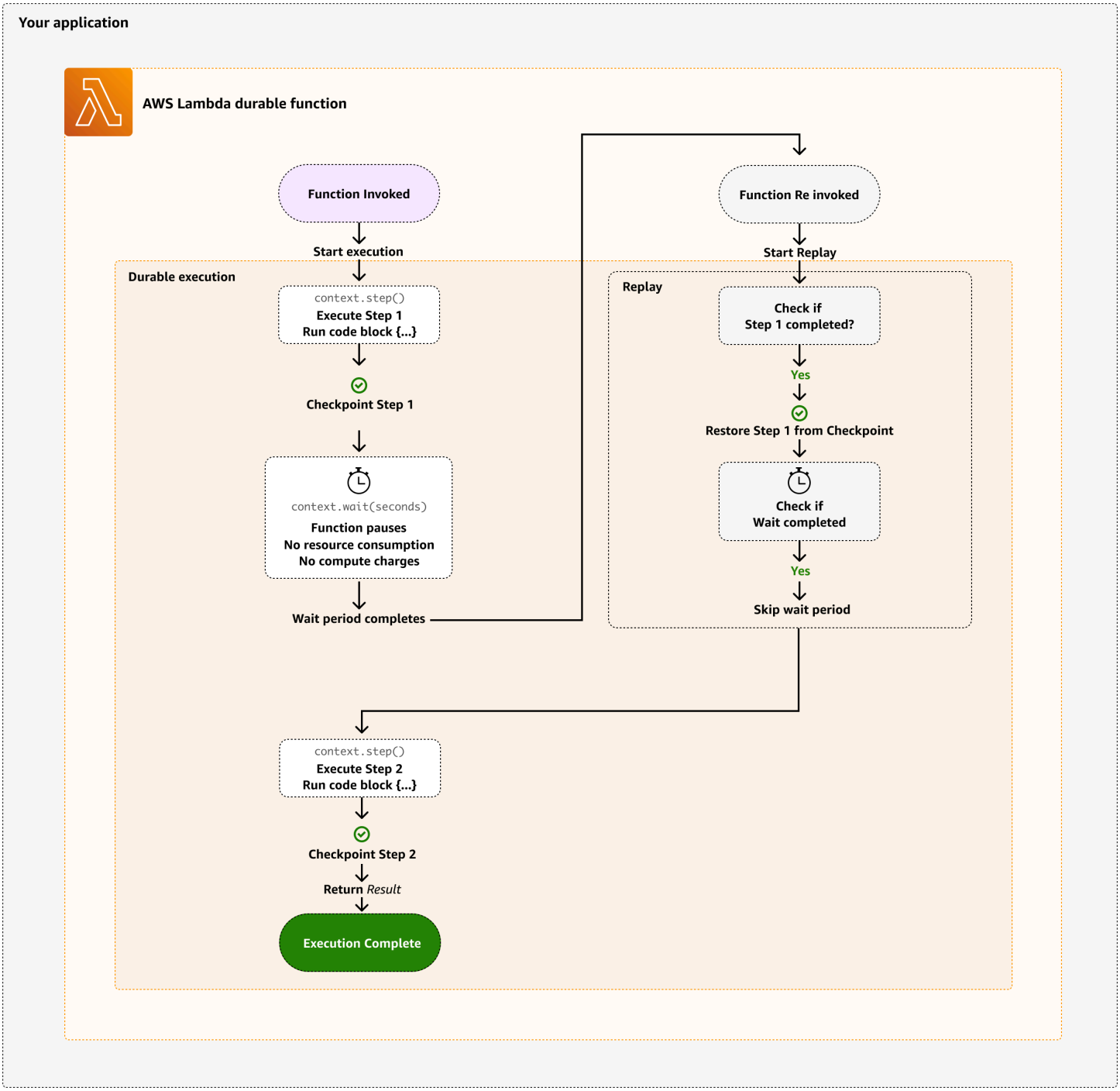
Cara Kerjanya

Di bawah kap, fungsi tahan lama adalah fungsi Lambda reguler menggunakan checkpoint/replay mekanisme untuk melacak kemajuan dan mendukung operasi yang berjalan lama melalui titik

suspensi yang ditentukan pengguna, yang biasa disebut sebagai eksekusi tahan lama. Setelah fungsi Anda dilanjutkan dari jeda atau gangguan, sistem melakukan pemutaran ulang. Selama pemutaran ulang, kode Anda berjalan dari awal tetapi melewati pos pemeriksaan yang telah selesai, menggunakan hasil yang disimpan alih-alih menjalankan kembali operasi yang telah selesai. Mekanisme replay ini memastikan konsistensi sambil memungkinkan eksekusi yang berjalan lama.

Untuk memanfaatkan mekanisme checkpoint-and-replay ini dalam aplikasi Anda, Lambda menyediakan SDK eksekusi yang tahan lama. SDK menghilangkan kompleksitas pengelolaan pos pemeriksaan dan pemutaran ulang, mengekspos primitif sederhana yang disebut operasi tahan lama yang Anda gunakan dalam kode Anda. SDK tersedia untuk JavaScript, TypeScript, Python, Java, Go, dan C # (.NET), terintegrasi mulus dengan alur kerja pengembangan Lambda Anda yang ada.

Dengan SDK, Anda membungkus event handler Lambda Anda, yang kemudian menyediakan DurableContext di samping acara Anda. Konteks ini memberi Anda akses ke operasi yang tahan lama seperti langkah dan penantian. Anda menulis logika fungsi Anda sebagai kode berurutan normal, tetapi alih-alih memanggil layanan secara langsung, Anda membungkus panggilan tersebut dalam langkah-langkah untuk pemeriksaan otomatis dan percobaan ulang. Ketika Anda perlu menjeda eksekusi, Anda menambahkan penantian yang menangguhkan fungsi Anda tanpa menimbulkan biaya. SDK menangani semua kompleksitas manajemen status dan pemutaran ulang di belakang layar, sehingga kode Anda tetap bersih dan dapat dibaca.



Langkah selanjutnya

- AWS Panduan Pengembang SDK Eksekusi Tahan Lama — Referensi SDK, permulaan cepat, pengujian, dan panduan bahasa
- Memulai dengan fungsi tahan lama
- Fungsi tahan lama atau Fungsi Langkah

- Pantau dan debug fungsi tahan lama
- Tinjau keamanan dan izin
- Mengenkripsi AWS Lambda data eksekusi tahan lama
- Ikuti praktik terbaik

Konsep Basic

The AWS Durable Execution SDK adalah fondasi untuk membangun fungsi yang tahan lama, menyediakan primitif yang Anda butuhkan untuk memeriksa kemajuan, menangani percobaan ulang, dan mengelola alur eksekusi. SDK tersedia untuk Node.js (JavaScript dan TypeScript), Python, Java, Go, dan C # (.NET). Untuk dokumentasi SDK lengkap termasuk referensi API, tutorial memulai cepat, dan panduan khusus bahasa, lihat AWS Panduan Pengembang SDK Eksekusi Tahan Lama.

Eksekusi tahan lama

SEBUAH eksekusi tahan lama mewakili siklus hidup lengkap fungsi tahan lama Lambda, menggunakan mekanisme checkpoint dan replay untuk melacak kemajuan logika bisnis, menangguhkan eksekusi, dan memulihkan dari kegagalan. Ketika fungsi dilanjutkan setelah penangguhan atau interupsi, pos pemeriksaan yang telah selesai sebelumnya diputar ulang dan fungsi melanjutkan eksekusi.

Siklus hidup mungkin mencakup beberapa pemanggilan fungsi Lambda untuk menyelesaikan eksekusi, terutama setelah penangguhan atau pemulihan kegagalan. Pendekatan ini memungkinkan fungsi Anda berjalan untuk waktu yang lama (hingga satu tahun) sambil mempertahankan kemajuan yang andal meskipun ada gangguan.

Cara kerja replay

Lambda menyimpan log yang berjalan dari semua operasi yang tahan lama (langkah, tunggu, dan operasi lainnya) saat fungsi Anda dijalankan. Ketika fungsi Anda perlu menjeda atau mengalami gangguan, Lambda menyimpan log pos pemeriksaan ini dan menghentikan eksekusi. Ketika tiba waktunya untuk melanjutkan, Lambda memanggil fungsi Anda lagi dari awal dan memutar ulang log pos pemeriksaan, menggantikan nilai yang disimpan untuk operasi yang diselesaikan. Ini berarti kode Anda berjalan lagi, tetapi langkah-langkah yang telah diselesaikan sebelumnya tidak dieksekusi ulang. Hasil yang disimpan digunakan sebagai gantinya.

Mekanisme replay ini sangat penting untuk memahami fungsi yang tahan lama. Kode Anda harus deterministik selama pemutaran ulang, artinya menghasilkan hasil yang sama dengan input yang

sama. Hindari operasi dengan efek samping (seperti menghasilkan angka acak atau mendapatkan waktu saat ini) di luar langkah, karena ini dapat menghasilkan nilai yang berbeda selama pemutaran ulang dan menyebabkan perilaku non-deterministik.

DurableContext

DurableContext adalah objek konteks yang diterima fungsi tahan lama Anda. Ini menyediakan metode untuk operasi yang tahan lama seperti langkah-langkah dan penantian yang membuat pos pemeriksaan dan mengelola aliran eksekusi.

Fungsi tahan lama Anda menerima `DurableContext` alih-alih konteks Lambda default:

TypeScript

```
import {
  DurableContext,
  withDurableExecution,
} from "@aws/durable-execution-sdk-js";

export const handler = withDurableExecution(
  async (event: any, context: DurableContext) => {
    const result = await context.step(async () => {
      return "step completed";
    });
    return result;
  },
);
```

Python

```
from aws_durable_execution_sdk_python import (
    DurableContext,
    durable_execution,
    durable_step,
)

@durable_step
def my_step(step_context, data):
    # Your business logic
    return result

@durable_execution
```

```
def handler(event, context: DurableContext):
    result = context.step(my_step(event["data"]))
    return result
```

Melalui `DurableContext`, fungsi Anda memiliki akses ke operasi tahan lama yang memeriksa kemajuan dan mengoordinasikan pekerjaan:

- Langkah-langkah jalankan logika bisnis dengan percobaan ulang bawaan dan checkpointing otomatis. Setiap langkah menyimpan hasilnya, sehingga fungsi Anda dilanjutkan dari langkah terakhir yang diselesaikan setelah gangguan.
- Menunggu adalah jeda yang direncanakan di mana fungsi Anda berhenti berjalan dan berhenti mengisi daya sampai saatnya untuk melanjutkan. Gunakan mereka untuk periode waktu, panggilan balik eksternal, atau polling untuk suatu kondisi.
- Memanggil memanggil fungsi Lambda lainnya, termasuk fungsi tahan lama, dan memeriksa hasilnya. Cross-account pemanggilan tidak didukung.

Untuk referensi bahasa lengkap untuk semua operasi tahan lama yang tersedia, lihat Referensi SDK di AWS Panduan Pengembang SDK Eksekusi Tahan Lama.

Konfigurasi fungsi tahan lama

Fungsi tahan lama menambahkan pengaturan konfigurasi yang mengontrol perilaku eksekusi dan retensi data. Pengaturan ini adalah tambahan untuk konfigurasi fungsi yang ada, seperti memori dan batas waktu fungsi. Mereka berlaku untuk seluruh eksekusi yang tahan lama daripada untuk pemanggilan individu.

DurableConfig objek mendefinisikan konfigurasi untuk fungsi tahan lama:

```
{
  "ExecutionTimeout": Integer,
  "RetentionPeriodInDays": Integer
}
```

Batas waktu pelaksanaan

batas waktu eksekusi mengontrol berapa lama eksekusi tahan lama dapat berjalan dari awal hingga selesai. Ini berbeda dari batas waktu fungsi Lambda, yang mengontrol berapa lama pemanggilan fungsi tunggal dapat berjalan.

Eksekusi yang tahan lama dapat mencakup beberapa pemanggilan fungsi Lambda saat berlangsung melalui pos pemeriksaan, tunggu, dan pemutaran ulang. Waktu tunggu eksekusi berlaku untuk total waktu yang telah berlalu dari eksekusi tahan lama, bukan untuk pemanggilan fungsi individu.

Memahami perbedaannya

Batas waktu fungsi membatasi setiap pemanggilan individual fungsi Anda, hingga 15 menit secara default atau 90 menit untuk pemanggilan asinkron dan pemetaan sumber peristiwa pada Instans Terkelola Lambda. Batas waktu eksekusi yang tahan lama (maksimum 1 tahun) membatasi total waktu dari saat eksekusi dimulai hingga selesai, gagal, atau batas waktu. Selama periode ini, fungsi Anda mungkin dipanggil beberapa kali saat memproses langkah-langkah, menunggu, dan pulih dari kegagalan.

Misalnya, jika Anda menetapkan batas waktu eksekusi tahan lama 24 jam dan batas waktu fungsi Lambda 5 menit:

- Setiap pemanggilan fungsi harus selesai dalam waktu 5 menit
- Seluruh eksekusi tahan lama dapat berjalan hingga 24 jam
- Fungsi Anda dapat dipanggil berkali-kali selama 24 jam tersebut
- Operasi tunggu tidak dihitung terhadap batas waktu fungsi Lambda tetapi dihitung terhadap batas waktu eksekusi

Anda dapat mengonfigurasi batas waktu eksekusi saat membuat fungsi tahan lama menggunakan konsol Lambda, AWS CLI, atau AWS SAM. Di konsol Lambda, pilih fungsi Anda, lalu Konfigurasi, eksekusi tahan lama. Atur nilai batas waktu eksekusi dalam detik (default: 86400 detik/24 jam, minimum: 60 detik, maksimum: 31536000 detik/1 tahun).

> **Note**
>
> Waktu tunggu eksekusi dan batas waktu fungsi adalah pengaturan yang berbeda. Batas waktu fungsi membatasi setiap pemanggilan, hingga 15 menit secara default atau 90 menit untuk pemanggilan pemetaan sumber asinkron dan peristiwa pada Instans Terkelola Lambda. Waktu tunggu eksekusi membatasi total waktu eksekusi tahan lama di semua pemanggilan (maksimum 1 tahun).

Periode retensi

periode retensi mengontrol berapa lama Lambda menyimpan riwayat eksekusi dan data pos pemeriksaan setelah eksekusi tahan lama selesai. Data ini mencakup hasil langkah, status eksekusi, dan log pos pemeriksaan lengkap.

Setelah periode retensi berakhir, Lambda menghapus riwayat eksekusi dan data pos pemeriksaan. Anda tidak dapat lagi mengambil detail eksekusi atau memutar ulang eksekusi. Periode retensi dimulai ketika eksekusi mencapai status terminal (SUCCEED, FAILED, STOPPED, atau TIMED_OUT).

Anda dapat mengonfigurasi periode retensi saat membuat fungsi tahan lama menggunakan konsol Lambda, AWS CLI, atau AWS SAM. Di konsol Lambda, pilih fungsi Anda, lalu Konfigurasi, eksekusi tahan lama. Tetapkan nilai periode retensi dalam hari (default: 14 hari, minimum: 1 hari, maksimum: 90 hari).

Pilih periode retensi berdasarkan persyaratan kepatuhan, kebutuhan debugging, dan pertimbangan biaya. Periode retensi yang lebih lama memberikan lebih banyak waktu untuk debugging dan audit tetapi meningkatkan biaya penyimpanan.

Lihat juga

- AWS Panduan Pengembang SDK Eksekusi Tahan Lama — Referensi SDK lengkap, tutorial memulai cepat, kerangka pengujian, dan panduan khusus bahasa.
- Fungsi tahan lama atau Fungsi Langkah — Bandingkan fungsi tahan lama dengan Fungsi Langkah untuk memahami kapan setiap pendekatan paling efektif.

Membuat fungsi Lambda yang tahan lama

Untuk memulai dengan fungsi Lambda yang tahan lama, gunakan konsol Lambda untuk membuat fungsi yang tahan lama. Dalam beberapa menit, Anda dapat membuat dan menerapkan fungsi tahan lama yang menggunakan langkah-langkah dan menunggu untuk menunjukkan eksekusi berbasis pos pemeriksaan.

Anda juga dapat membuat fungsi tahan lama menggunakan AWS CLIor infrastructure as code (IAc).

Saat Anda menjalankan tutorial, Anda akan mempelajari konsep fungsi dasar yang tahan lama, seperti cara menggunakan `DurableContext` objek, membuat pos pemeriksaan dengan langkah-

langkah, dan menjeda eksekusi dengan menunggu. Anda juga akan mempelajari cara kerja replay saat fungsi Anda dilanjutkan setelah menunggu.

Untuk menjaga hal-hal sederhana, tutorial ini menunjukkan kepada Anda cara membuat fungsi Anda menggunakan Python atau Node.js runtime. Dengan bahasa yang ditafsirkan ini, Anda dapat mengedit kode fungsi langsung di editor kode bawaan konsol.

> **Note**
>
> Fungsi tahan lama saat ini mendukung runtime Python, Node.js (JavaScript/TypeScript), dan Java, serta gambar kontainer (OCI). Untuk daftar lengkap versi runtime yang didukung dan opsi gambar kontainer, lihat Runtime yang didukung untuk fungsi tahan lama. Untuk informasi selengkapnya tentang penggunaan gambar kontainer dengan Lambda, lihat Membuat gambar kontainer Lambda di Panduan Pengembang Lambda.

> **Tip**
>
> Untuk mempelajari cara membuat solusi tanpa server, lihat Panduan Pengembang Tanpa Server.

Buat fungsi Lambda yang tahan lama dengan konsol

Dalam contoh ini, fungsi tahan lama Anda memproses pesanan melalui beberapa langkah dengan pos pemeriksaan otomatis. Fungsi mengambil objek JSON yang berisi ID pesanan, memvalidasi pesanan, memproses pembayaran, dan mengonfirmasi pesanan. Setiap langkah secara otomatis diperiksa, jadi jika fungsi terganggu, itu dilanjutkan dari langkah terakhir yang diselesaikan.

Fungsi Anda juga menunjukkan operasi tunggu, menghentikan eksekusi untuk waktu yang singkat untuk mensimulasikan menunggu konfirmasi eksternal.

Untuk membuat fungsi yang tahan lama dengan konsol

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih Buat Fungsi Acara.
3. Pilih Penulis dari awal.
4. Di panel Informasi dasar, untuk nama Fungsi, masukkan**`myDurableFunction`**.

5. Untuk Runtime, pilih Node.js 24 atau Python 3.14.
6. Pilih Aktifkan eksekusi yang tahan lama.

Lambda membuat fungsi tahan lama Anda dengan peran eksekusi yang menyertakan izin untuk operasi pos pemeriksaan (dan). `lambda:CheckpointDurableExecution` `lambda:GetDurableExecutionState`

> **Note**
>
> Runtime Lambda menyertakan Durable Execution SDK, sehingga Anda dapat menguji fungsi yang tahan lama tanpa dependensi kemasan. Namun, sebaiknya sertakan SDK dalam paket penerapan Anda untuk produksi. Ini memastikan konsistensi versi dan menghindari kemungkinan pembaruan runtime yang mungkin memengaruhi fungsi Anda.

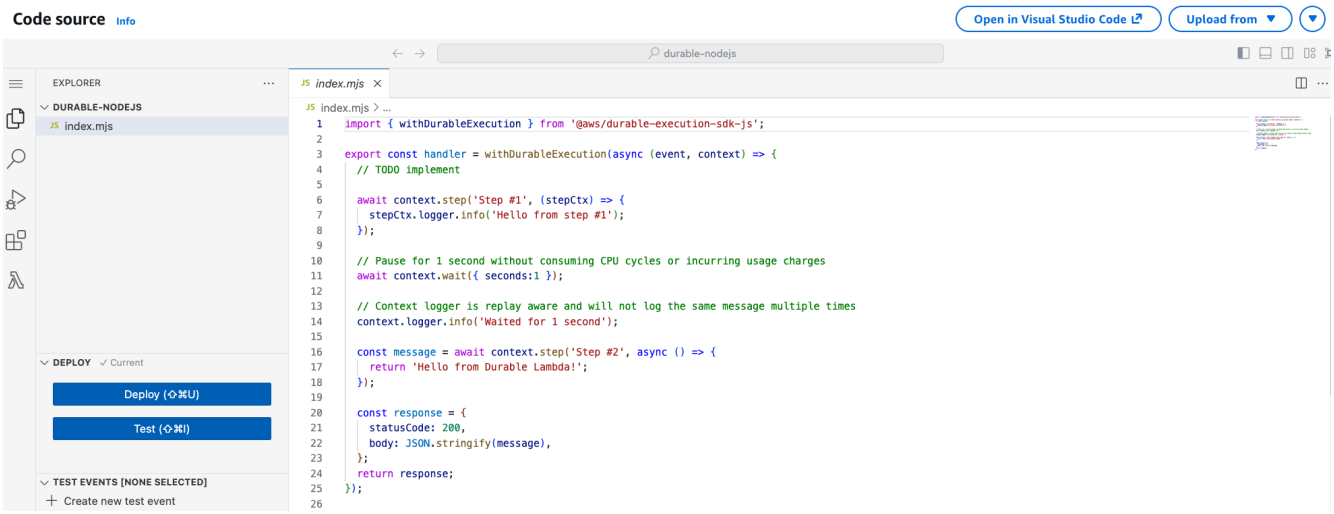
Gunakan editor kode bawaan konsol untuk menambahkan kode fungsi tahan lama Anda.

Node.js

Untuk memodifikasi kode di konsol

1. Pilih tab Kode.

   Di editor kode bawaan konsol, Anda akan melihat kode fungsi yang dibuat Lambda. Jika Anda tidak melihat tab index.mjs di editor kode, pilih index.mjs di file explorer seperti yang ditunjukkan pada diagram berikut.



2. Tempelkan kode berikut ke tab index.mjs, ganti kode yang dibuat Lambda.

```
import {
  withDurableExecution,
} from "@aws/durable-execution-sdk-js";

export const handler = withDurableExecution(
  async (event, context) => {
    const orderId = event.orderId;

    // Step 1: Validate order
    const validationResult = await context.step(async (stepContext) => {
      stepContext.logger.info(`Validating order ${orderId}`);
      return { orderId, status: "validated" };
    });

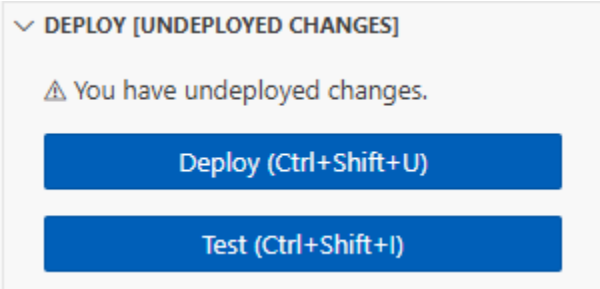
    // Step 2: Process payment
    const paymentResult = await context.step(async (stepContext) => {
      stepContext.logger.info(`Processing payment for order ${orderId}`);
      return { orderId, status: "paid", amount: 99.99 };
    });

    // Wait for 10 seconds to simulate external confirmation
    await context.wait({ seconds: 10 });

    // Step 3: Confirm order
    const confirmationResult = await context.step(async (stepContext) => {
      stepContext.logger.info(`Confirming order ${orderId}`);
      return { orderId, status: "confirmed" };
    });

    return {
      orderId: orderId,
      status: "completed",
      steps: [validationResult, paymentResult, confirmationResult]
    };
  }
);
```

3. Di bagian DEPLOY, pilih Deploy untuk memperbarui kode fungsi Anda:



Memahami kode fungsi tahan lama Anda

Sebelum Anda pindah ke langkah berikutnya, mari kita lihat kode fungsi dan pahami konsep fungsi kunci yang tahan lama.

- `withDurableExecution`Pembungkusnya:

  Fungsi tahan lama Anda dibungkus dengan`withDurableExecution`. Pembungkus ini memungkinkan eksekusi yang tahan lama dengan menyediakan `DurableContext` objek dan mengelola operasi pos pemeriksaan.

- `DurableContext`Objek:

  Alih-alih konteks Lambda standar, fungsi Anda menerima file. `DurableContext` Objek ini menyediakan metode untuk operasi tahan lama seperti `step()` dan `wait()` yang membuat pos pemeriksaan.

- Langkah-langkah dan pos pemeriksaan:

  Setiap `context.step()` panggilan membuat pos pemeriksaan sebelum dan sesudah eksekusi. Jika fungsi Anda terganggu, itu dilanjutkan dari pos pemeriksaan terakhir yang selesai. Fungsi ini tidak menjalankan kembali langkah-langkah yang telah selesai. Ini menggunakan hasil yang disimpan sebagai gantinya.

- Tunggu operasi:

  `context.wait()`Panggilan menjeda eksekusi tanpa menghabiskan sumber daya komputasi. Ketika menunggu selesai, Lambda memanggil fungsi Anda lagi dan memutar ulang log pos pemeriksaan, menggantikan nilai yang disimpan untuk langkah-langkah yang diselesaikan.

- Mekanisme replay:

  Ketika fungsi Anda dilanjutkan setelah menunggu atau interupsi, Lambda menjalankan kode Anda dari awal. Namun, langkah yang diselesaikan tidak dijalankan ulang. Lambda memutar

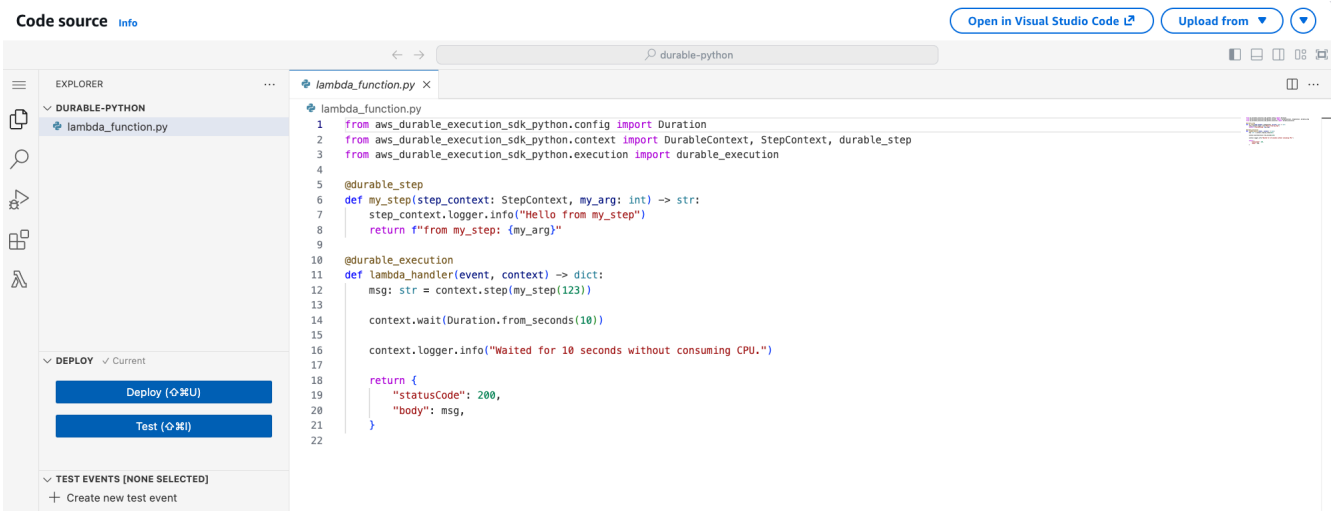
ulang hasil mereka dari log pos pemeriksaan. Inilah sebabnya mengapa kode Anda harus deterministik.

Python

Untuk memodifikasi kode di konsol

1. Pilih tab Kode.

   Di editor kode bawaan konsol, Anda akan melihat kode fungsi yang dibuat Lambda. Jika Anda tidak melihat tab lambda_function.py di editor kode, pilih lambda_function.py di file explorer seperti yang ditunjukkan pada diagram berikut.



2. Tempelkan kode berikut ke tab lambda_function.py, ganti kode yang dibuat Lambda.

```
from aws_durable_execution_sdk_python import (
    DurableContext,
    durable_execution,
    durable_step,
)
from aws_durable_execution_sdk_python.config import Duration

@durable_step
def validate_order(step_context, order_id):
    step_context.logger.info(f"Validating order {order_id}")
    return {"orderId": order_id, "status": "validated"}

@durable_step
def process_payment(step_context, order_id):
```

```
    step_context.logger.info(f"Processing payment for order {order_id}")
    return {"orderId": order_id, "status": "paid", "amount": 99.99}

@durable_step
def confirm_order(step_context, order_id):
    step_context.logger.info(f"Confirming order {order_id}")
    return {"orderId": order_id, "status": "confirmed"}

@durable_execution
def lambda_handler(event, context: DurableContext):
    order_id = event['orderId']

    # Step 1: Validate order
    validation_result = context.step(validate_order(order_id))

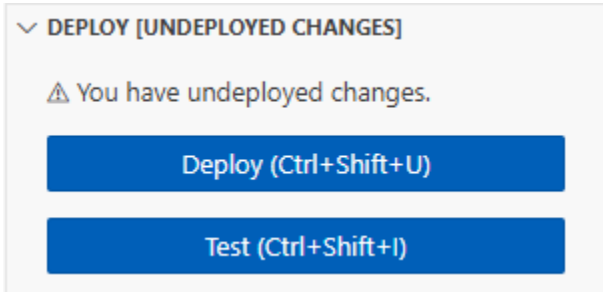
    # Step 2: Process payment
    payment_result = context.step(process_payment(order_id))

    # Wait for 10 seconds to simulate external confirmation
    context.wait(Duration.from_seconds(10))

    # Step 3: Confirm order
    confirmation_result = context.step(confirm_order(order_id))

    return {
        "orderId": order_id,
        "status": "completed",
        "steps": [validation_result, payment_result, confirmation_result]
    }
```

3. Di bagian DEPLOY, pilih Deploy untuk memperbarui kode fungsi Anda:



Memahami kode fungsi tahan lama Anda

Sebelum Anda pindah ke langkah berikutnya, mari kita lihat kode fungsi dan pahami konsep fungsi kunci yang tahan lama.

- `@durable_execution`Dekorator:

  Fungsi handler Anda dihiasi dengan`@durable_execution`. Dekorator ini memungkinkan eksekusi yang tahan lama dengan menyediakan `DurableContext` objek dan mengelola operasi pos pemeriksaan.
- `@durable_step`Dekorator:

  Setiap fungsi langkah dihiasi dengan`@durable_step`. Dekorator ini menandai fungsi sebagai langkah tahan lama yang menciptakan pos pemeriksaan.
- `DurableContext`Objek:

  Alih-alih konteks Lambda standar, fungsi Anda menerima file. `DurableContext` Objek ini menyediakan metode untuk operasi tahan lama seperti `step()` dan `wait()` yang membuat pos pemeriksaan.
- Langkah-langkah dan pos pemeriksaan:

  Setiap `context.step()` panggilan membuat pos pemeriksaan sebelum dan sesudah eksekusi. Jika fungsi Anda terganggu, itu dilanjutkan dari pos pemeriksaan terakhir yang selesai. Fungsi ini tidak menjalankan kembali langkah-langkah yang telah selesai. Ini menggunakan hasil yang disimpan sebagai gantinya.
- Tunggu operasi:

  `context.wait()`Panggilan menjeda eksekusi tanpa menghabiskan sumber daya komputasi. Ketika menunggu selesai, Lambda memanggil fungsi Anda lagi dan memutar ulang log pos pemeriksaan, menggantikan nilai yang disimpan untuk langkah-langkah yang diselesaikan.
- Python SDK sinkron:

  Perhatikan bahwa Python SDK tidak digunakan. `await` Semua operasi tahan lama adalah panggilan metode sinkron.

Memanggil fungsi tahan lama menggunakan editor kode konsol

Jika tidak ada versi eksplisit yang ditentukan (atau dipublikasikan), konsol akan memanggil fungsi tahan lama menggunakan kualifikasi `$LATEST` versi. Namun, untuk eksekusi deterministik kode Anda, Anda harus selalu menggunakan ARN yang memenuhi syarat yang menunjuk ke versi stabil.

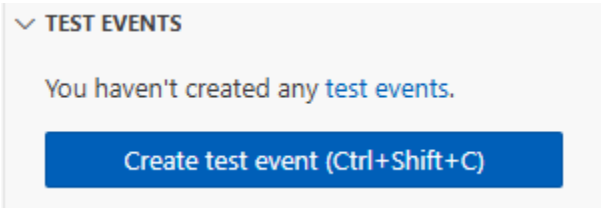
Untuk mempublikasikan versi fungsi Anda

1. Pilih tab Versi.
2. PilihPublikasikan versi baru.
3. Untuk deskripsi Versi, masukkan **`Initial version`** (opsional).
4. Pilih Terbitkan.
5. Lambda membuat versi 1 dari fungsi Anda. Perhatikan bahwa fungsi ARN sekarang termasuk `:1` di akhir, menunjukkan ini adalah versi 1.

Sekarang buat acara uji untuk dikirim ke fungsi Anda. Acara ini adalah dokumen berformat JSON yang berisi ID pesanan.

Untuk membuat acara pengujian

1. Di bagian TEST EVENTS pada editor kode konsol, pilih Buat acara pengujian.



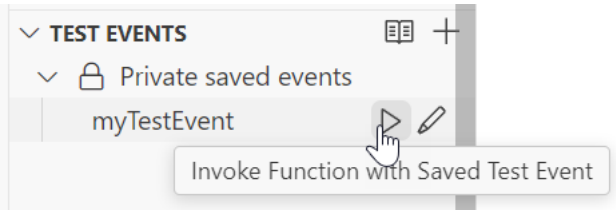
2. Untuk Nama Acara, masukkan**`myTestEvent`**.
3. Di bagian Event JSON, ganti JSON default dengan yang berikut:

```
{
  "orderId": "order-12345"
}
```

4. Pilih Simpan.

Untuk menguji fungsi tahan lama Anda dan melihat eksekusi

Di bagian TEST EVENTS pada editor kode konsol, pilih ikon jalankan di sebelah acara pengujian Anda:



Fungsi tahan lama Anda mulai dijalankan. Karena ini termasuk menunggu 10 detik, pemanggilan awal selesai dengan cepat, dan fungsi dilanjutkan setelah periode tunggu. Anda dapat melihat progres eksekusi di tab Eksekusi tahan lama.

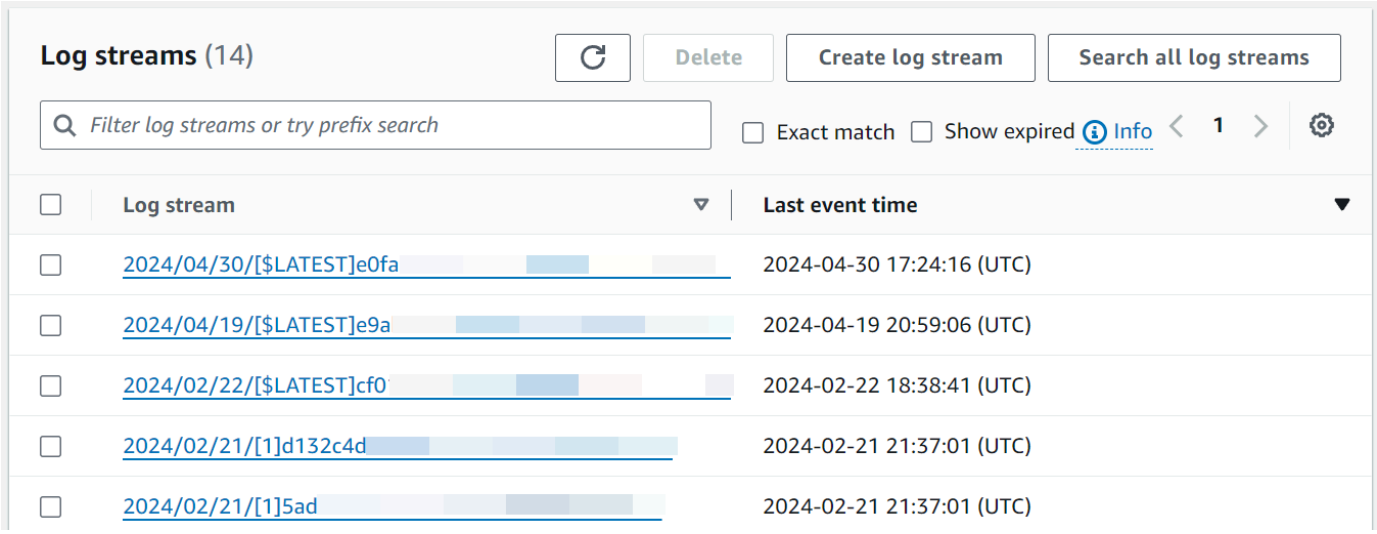
Untuk melihat eksekusi fungsi tahan lama

1. Pilih tab Eksekusi tahan lama.
2. Temukan eksekusi Anda dalam daftar. Eksekusi menunjukkan status saat ini (Berjalan, Berhasil, atau Gagal).
3. Pilih ID eksekusi untuk melihat detail, termasuk:
   - Garis waktu eksekusi menunjukkan kapan setiap langkah selesai
   - Sejarah pos pemeriksaan
   - Periode tunggu
   - Hasil langkah

Anda juga dapat melihat log fungsi Anda di CloudWatch Log untuk melihat keluaran konsol dari setiap langkah.

Untuk melihat catatan pemanggilan fungsi Anda di Log CloudWatch

1. Buka halaman Grup log CloudWatch konsol.
2. Pilih grup log untuk fungsi Anda (`/aws/lambda/myDurableFunction`).
3. Gulir ke bawah dan pilih aliran Log untuk pemanggilan fungsi yang ingin Anda lihat.



Anda akan melihat entri log untuk setiap pemanggilan fungsi Anda, termasuk eksekusi awal dan pemutaran ulang setelah menunggu.

> **Note**
>
> Saat Anda menggunakan logger dari `DurableContext` (seperti `context.logger` atau`stepContext.logger`), log juga muncul dalam eksekusi tahan lama dan tampilan langkah di konsol Lambda. Log ini mungkin membutuhkan waktu beberapa saat untuk dimuat.

Bersihkan

Setelah Anda selesai bekerja dengan contoh fungsi tahan lama, hapus. Anda juga dapat menghapus grup log yang menyimpan log fungsi, dan peran eksekusi yang dibuat konsol.

Untuk menghapus Fungsi Acara Lambda

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih Fungsi Acara yang Anda buat.
3. Pilih Tindakan, Hapus.
4. Ketik **`confirm`** kolom input teks dan pilih Hapus.

Untuk menghapus grup log

1. Buka halaman Grup log CloudWatch konsol.
2. Pilih grup log fungsi (`/aws/lambda/myDurableFunction`).
3. Pilih Tindakan, Hapus grup log.
4. Di kotak dialog Hapus grup log, pilih Hapus.

Untuk menghapus peran eksekusi

1. Buka halaman Peran konsol AWS Identity and Access Management (IAM).
2. Pilih peran eksekusi fungsi (misalnya,`myDurableFunction-role-31exxmpl`).
3. Pilih Hapus.
4. Di kotak dialog Hapus peran, masukkan nama peran, lalu pilih Hapus.

Sumber daya tambahan dan langkah selanjutnya

Sekarang setelah Anda membuat dan menguji fungsi tahan lama sederhana menggunakan konsol, ambil langkah berikut:

- Pelajari tentang kasus penggunaan umum untuk fungsi tahan lama, termasuk transaksi terdistribusi, pemrosesan pesanan, dan alur kerja tinjauan manusia. Lihat Contoh.
- Pahami cara memantau eksekusi fungsi yang tahan lama dengan CloudWatch metrik dan riwayat eksekusi. Lihat Monitoring dan debugging.
- Pelajari tentang menjalankan fungsi tahan lama secara sinkron dan asinkron, serta mengelola eksekusi yang berjalan lama. Lihat Memanggil fungsi yang tahan lama.
- Ikuti praktik terbaik untuk menulis kode deterministik, mengelola ukuran pos pemeriksaan, dan mengoptimalkan biaya. Lihat Praktik terbaik.
- Pelajari cara menguji fungsi tahan lama secara lokal dan di cloud. Lihat Menguji fungsi tahan lama.
- Bandingkan fungsi tahan lama dengan Step Functions untuk memahami kapan setiap pendekatan paling efektif. Lihat Fungsi tahan lama atau Step Functions.

Menyebarkan dan menjalankan fungsi Lambda yang tahan lama dengan AWS CLI

Gunakan AWS CLI untuk membuat dan menerapkan fungsi tahan lama Lambda dengan perintah imperatif. Pendekatan ini memberi Anda kontrol langsung atas setiap langkah proses penyebaran.

Prasyarat

- Instal dan konfigurasikan AWS CLI. Untuk petunjuk, lihat Menginstal AWS CLI.
- Buat paket penerapan dengan kode fungsi Anda dan SDK eksekusi yang tahan lama.
- Buat peran eksekusi IAM dengan izin pos pemeriksaan.

Buat peran eksekusi

Buat peran IAM dengan izin untuk eksekusi Lambda dasar dan operasi pos pemeriksaan.

Untuk membuat peran eksekusi

1. Buat dokumen kebijakan kepercayaan yang memungkinkan Lambda untuk mengambil peran. Simpan ini sebagai`trust-policy.json`:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "lambda.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

2. Buat peran:

```
aws iam create-role \
  --role-name durable-function-role \
  --assume-role-policy-document file://trust-policy.json
```

3. Lampirkan kebijakan eksekusi tahan lama untuk operasi pos pemeriksaan dan eksekusi dasar:

```
aws iam attach-role-policy \
  --role-name durable-function-role \
  --policy-arn arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicDurableExecutionRolePolicy
```

Kebijakan `AWSLambdaBasicDurableExecutionRolePolicy` terkelola mencakup izin yang diperlukan untuk operasi pos pemeriksaan (`lambda:CheckpointDurableExecution`dan`lambda:GetDurableExecutionState`) dan eksekusi Lambda dasar.

Buat fungsi yang tahan lama

Buat fungsi tahan lama Anda dengan `--durable-config` parameter.

Untuk membuat fungsi yang tahan lama

1. Package kode fungsi Anda dengan dependensi ke dalam file.zip:

```
zip -r function.zip index.mjs node_modules/
```

> **Note**
>
> Untuk fungsi tahan lama berbasis Java, Anda perlu mengkompilasi kode fungsi dan dependensi Anda ke dalam satu file.zip atau file Java Archive (JAR). Untuk informasi selengkapnya, lihat Menerapkan fungsi Java Lambda dengan arsip file.zip atau JAR.

2. Buat fungsi dengan eksekusi tahan lama diaktifkan:

```
aws lambda create-function \
  --function-name myDurableFunction \
  --runtime nodejs22.x \
  --role arn:aws:iam::123456789012:role/durable-function-role \
  --handler index.handler \
  --zip-file fileb://function.zip \
  --durable-config '{"ExecutionTimeout": 3600, "RetentionPeriodInDays": 7}'
```

> **Note**
>
> Anda hanya dapat mengaktifkan eksekusi yang tahan lama saat membuat fungsi. Anda tidak dapat mengaktifkannya pada fungsi yang ada.

Publikasikan versi

Meskipun fungsi tahan lama dapat dipanggil menggunakan kualifikasi `$LATEST` versi, Anda harus selalu menggunakan ARN yang memenuhi syarat yang menunjuk ke versi stabil untuk memastikan eksekusi kode Anda secara deterministik.

```
aws lambda publish-version \
  --function-name myDurableFunction \
  --description "Initial version"
```

Perintah mengembalikan versi ARN. Perhatikan nomor versi (misalnya, `:1`) di akhir ARN.

Secara opsional, buat alias yang menunjuk ke versi:

```
aws lambda create-alias \
  --function-name myDurableFunction \
  --name prod \
  --function-version 1
```

Memanggil fungsi tahan lama

Panggil fungsi tahan lama Anda menggunakan ARN (versi atau alias) yang memenuhi syarat.

> **Note**
>
> Pemanggilan idempoten: Untuk mencegah eksekusi duplikat saat mencoba kembali pemanggilan yang gagal, Anda dapat memberikan nama eksekusi yang memastikan semantik eksekusi. at-most-once Lihat Idempotency untuk detailnya.

Invokasi sinkron

Untuk eksekusi yang selesai dalam 15 menit, gunakan pemanggilan sinkron:

```
aws lambda invoke \
  --function-name myDurableFunction:1 \
  --payload '{"orderId": "order-12345"}' \
  --cli-binary-format raw-in-base64-out \
  response.json
```

Atau menggunakan alias:

```
aws lambda invoke \
  --function-name myDurableFunction:prod \
  --payload '{"orderId": "order-12345"}' \
  --cli-binary-format raw-in-base64-out \
  response.json
```

Invokasi asinkron

Untuk eksekusi yang berjalan lama, gunakan pemanggilan asinkron:

```
aws lambda invoke \
  --function-name myDurableFunction:prod \
  --invocation-type Event \
  --payload '{"orderId": "order-12345"}' \
  --cli-binary-format raw-in-base64-out \
  response.json
```

Dengan pemanggilan asinkron, Lambda segera kembali. Fungsi terus dijalankan di latar belakang.

> **Note**
>
> Anda dapat menggunakan `$LATEST` untuk membuat prototipe dan pengujian di konsol. Untuk beban kerja produksi, gunakan versi atau alias yang diterbitkan.

Kelola eksekusi yang tahan lama

Gunakan perintah berikut untuk mengelola dan memantau eksekusi fungsi yang tahan lama.

Daftar eksekusi

Daftar semua eksekusi untuk fungsi yang tahan lama:

```
aws lambda list-durable-executions-by-function \
  --function-name myDurableFunction
```

Dapatkan detail eksekusi

Dapatkan detail tentang eksekusi tertentu:

```
aws lambda get-durable-execution \
  --durable-execution-arn arn:aws:lambda:us-
east-1:123456789012:function:myDurableFunction:my-function-version/durable-execution/
my-execution-name/my-execution-id
```

Dapatkan riwayat eksekusi

Lihat riwayat pos pemeriksaan untuk eksekusi:

```
aws lambda get-durable-execution-history \
  --durable-execution-arn arn:aws:lambda:us-
east-1:123456789012:function:myDurableFunction:my-function-version/durable-execution/
my-execution-name/my-execution-id
```

Hentikan eksekusi

Hentikan eksekusi tahan lama yang berjalan:

```
aws lambda stop-durable-execution \
  --durable-execution-arn arn:aws:lambda:us-
east-1:123456789012:function:myDurableFunction:my-function-version/durable-execution/
my-execution-name/my-execution-id
```

Perbarui kode fungsi

Perbarui kode fungsi tahan lama Anda dan publikasikan versi baru:

Untuk memperbarui dan mempublikasikan versi baru

1. Perbarui kode fungsi:

   ```
   aws lambda update-function-code \
     --function-name myDurableFunction \
     --zip-file fileb://function.zip
   ```

2. Tunggu pembaruan selesai:

   ```
   aws lambda wait function-updated \
   ```

```
  --function-name myDurableFunction
```

3. Publikasikan versi baru:

```
aws lambda publish-version \
  --function-name myDurableFunction \
  --description "Updated order processing logic"
```

4. Perbarui alias untuk menunjuk ke versi baru:

```
aws lambda update-alias \
  --function-name myDurableFunction \
  --name prod \
  --function-version 2
```

> **Important**
>
> Menjalankan eksekusi terus menggunakan versi yang mereka mulai. Pemanggilan baru menggunakan versi alias yang diperbarui.

Lihat log fungsi

Lihat log fungsi tahan lama Anda di CloudWatch Log:

```
aws logs tail /aws/lambda/myDurableFunction --follow
```

Filter log untuk eksekusi tertentu:

```
aws logs filter-log-events \
  --log-group-name /aws/lambda/myDurableFunction \
  --filter-pattern "exec-abc123"
```

Pembersihan sumber daya

Hapus fungsi tahan lama dan sumber daya terkait:

```
# Delete the function
aws lambda delete-function --function-name myDurableFunction
```

```
# Delete the IAM role policies
aws iam detach-role-policy \
  --role-name durable-function-role \
  --policy-arn arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole

aws iam detach-role-policy \
  --role-name durable-function-role \
  --policy-arn arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicDurableExecutionRolePolicy

# Delete the role
aws iam delete-role --role-name durable-function-role
```

Langkah selanjutnya

Setelah menerapkan fungsi tahan lama Anda dengan: AWS CLI

- Memantau eksekusi menggunakan perintah `list-durable-executions-by-function` dan `get-durable-execution`
- Lihat operasi pos pemeriksaan dalam peristiwa AWS CloudTrail data
- Siapkan CloudWatch alarm untuk kegagalan eksekusi atau eksekusi yang berjalan lama
- Mengotomatiskan penerapan menggunakan skrip shell atau pipeline CI/CD

Untuk informasi selengkapnya tentang AWS CLI perintah untuk Lambda, lihat Referensi AWS CLI Perintah.

Terapkan fungsi tahan lama Lambda dengan Infrastructure as Code

Anda dapat menerapkan fungsi tahan lama Lambda menggunakan alat Infrastructure as Code (IAc) seperti AWS CloudFormation,, AWS CDK AWS Serverless Application Model, atau Terraform. Alat ini memungkinkan Anda menentukan fungsi, peran eksekusi, dan izin dalam kode, membuat penerapan dapat diulang dan dikendalikan versi.

Ketiga alat mengharuskan Anda untuk:

- Aktifkan eksekusi tahan lama pada fungsi
- Berikan izin pos pemeriksaan ke peran eksekusi
- Publikasikan versi atau buat alias (fungsi tahan lama memerlukan ARN yang memenuhi syarat)

Fungsi tahan lama dari ZIP

AWS CloudFormation

Gunakan CloudFormation untuk menentukan fungsi tahan lama Anda dalam template. Contoh berikut menciptakan fungsi tahan lama dengan izin yang diperlukan.

```
AWSTemplateFormatVersion: '2010-09-09'
Description: Lambda durable function example

Resources:
  DurableFunctionRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Principal:
              Service: lambda.amazonaws.com
            Action: sts:AssumeRole
      ManagedPolicyArns:
        - arn:aws:iam::aws:policy/service-role/AWSLambdaBasicDurableExecutionRolePolicy

  DurableFunction:
    Type: AWS::Lambda::Function
    Properties:
      FunctionName: myDurableFunction
      Runtime: nodejs22.x
      Handler: index.handler
      Role: !GetAtt DurableFunctionRole.Arn
      Code:
        ZipFile: |
          // Your durable function code here
          export const handler = async (event, context) => {
            return { statusCode: 200 };
          };
      DurableConfig:
        ExecutionTimeout: 3600
        RetentionPeriodInDays: 7

  DurableFunctionVersion:
    Type: AWS::Lambda::Version
    Properties:
```

```
      FunctionName: !Ref DurableFunction
      Description: Initial version

  DurableFunctionAlias:
    Type: AWS::Lambda::Alias
    Properties:
      FunctionName: !Ref DurableFunction
      FunctionVersion: !GetAtt DurableFunctionVersion.Version
      Name: prod

Outputs:
  FunctionArn:
    Description: Durable function ARN
    Value: !GetAtt DurableFunction.Arn
  AliasArn:
    Description: Function alias ARN (use this for invocations)
    Value: !Ref DurableFunctionAlias
```

Untuk menerapkan template

```
aws cloudformation deploy \
  --template-file template.yaml \
  --stack-name my-durable-function-stack \
  --capabilities CAPABILITY_IAM
```

AWS CDK

Dengan AWS CDK, Anda dapat mendefinisikan infrastruktur menggunakan bahasa pemrograman. Contoh berikut menunjukkan cara membuat fungsi tahan lama menggunakan TypeScript dan Python.

TypeScript

```
import * as cdk from 'aws-cdk-lib';
import * as lambda from 'aws-cdk-lib/aws-lambda';
import * as iam from 'aws-cdk-lib/aws-iam';
import { Construct } from 'constructs';

export class DurableFunctionStack extends cdk.Stack {
  constructor(scope: Construct, id: string, props?: cdk.StackProps) {
    super(scope, id, props);

    // Create the durable function
    const durableFunction = new lambda.Function(this, 'DurableFunction', {
```

```
      runtime: lambda.Runtime.NODEJS_22_X,
      handler: 'index.handler',
      code: lambda.Code.fromAsset('lambda'),
      functionName: 'myDurableFunction',
      durableConfig: { executionTimeout: Duration.hours(1), retentionPeriod:
 Duration.days(30) },
    });

    // Create version and alias
    const version = durableFunction.currentVersion;
    const alias = new lambda.Alias(this, 'ProdAlias', {
      aliasName: 'prod',
      version: version,
    });

    // Output the alias ARN
    new cdk.CfnOutput(this, 'FunctionAliasArn', {
      value: alias.functionArn,
      description: 'Use this ARN to invoke the durable function',
    });
  }
}
```

Python

```
from aws_cdk import (
    Stack,
    aws_lambda as lambda_,
    aws_iam as iam,
    CfnOutput,
)
from constructs import Construct

class DurableFunctionStack(Stack):
    def __init__(self, scope: Construct, id: str, **kwargs):
        super().__init__(scope, id, **kwargs)

        # Create the durable function
        durable_function = lambda_.Function(
            self, 'DurableFunction',
            runtime=lambda_.Runtime.NODEJS_22_X,
            handler='index.handler',
            code=lambda_.Code.from_asset('lambda'),
```

```
            function_name='myDurableFunction',
            durable_execution={execution_timeout: Duration.hours(1),
 retention_period: Duration.days(30)}
        )

        # Add durable execution managed policy for checkpoint permissions
        durable_function.role.add_managed_policy(
            iam.ManagedPolicy.from_aws_managed_policy_name('service-role/
AWSLambdaBasicDurableExecutionRolePolicy')
        )

        # Create version and alias
        version = durable_function.current_version
        alias = lambda_.Alias(
            self, 'ProdAlias',
            alias_name='prod',
            version=version
        )

        # Output the alias ARN
        CfnOutput(
            self, 'FunctionAliasArn',
            value=alias.function_arn,
            description='Use this ARN to invoke the durable function'
        )
```

Untuk menerapkan tumpukan CDK

```
cdk deploy
```

AWS Serverless Application Model

AWS SAM menyederhanakan CloudFormation template untuk aplikasi tanpa server. Template berikut menciptakan fungsi tahan lama dengan AWS SAM.

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: Lambda durable function with SAM

Resources:
  DurableFunction:
    Type: AWS::Serverless::Function
```

```
    Properties:
      FunctionName: myDurableFunction
      Runtime: nodejs22.x
      Handler: index.handler
      CodeUri: ./src
      DurableConfig:
        ExecutionTimeout: 3600
        RetentionPeriodInDays: 7
      Policies:
        - arn:aws:iam::aws:policy/service-role/AWSLambdaBasicDurableExecutionRolePolicy
      AutoPublishAlias: prod

Outputs:
  FunctionArn:
    Description: Durable function ARN
    Value: !GetAtt DurableFunction.Arn
  AliasArn:
    Description: Function alias ARN (use this for invocations)
    Value: !Ref DurableFunction.Alias
```

Untuk menerapkan template SAM

```
sam build
sam deploy --guided
```

Terraform

Terraform adalah alat IaC open-source populer yang mendukung sumber daya. AWS Contoh berikut membuat fungsi tahan lama dengan Terraform menggunakan AWS penyedia versi 6.25.0 atau yang lebih baru.

```
terraform {
  required_version = ">= 1.0"
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = ">= 6.25.0"
    }
  }
}

provider "aws" {
  region = "us-east-2"
```

```
}

# IAM Role for Lambda Function
resource "aws_iam_role" "lambda_role" {
  name = "durable-function-role"

  assume_role_policy = jsonencode({
    Version = "2012-10-17"
    Statement = [{
      Action = "sts:AssumeRole"
      Effect = "Allow"
      Principal = {
        Service = "lambda.amazonaws.com"
      }
    }]
  })
}

# Attach durable execution policy for checkpoint operations
resource "aws_iam_role_policy_attachment" "lambda_durable" {
  policy_arn = "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicDurableExecutionRolePolicy"
  role       = aws_iam_role.lambda_role.name
}

# Lambda Function with Durable Execution enabled
resource "aws_lambda_function" "durable_function" {
  filename      = "function.zip"
  function_name = "myDurableFunction"
  role          = aws_iam_role.lambda_role.arn
  handler       = "index.handler"
  runtime       = "nodejs22.x"
  timeout       = 30
  memory_size   = 512

  durable_config {
    execution_timeout = 900
    retention_period  = 7
  }
}

# Publish a version
resource "aws_lambda_alias" "prod" {
  name             = "prod"
```

```
  function_name    = aws_lambda_function.durable_function.function_name
  function_version = aws_lambda_function.durable_function.version
}

output "function_arn" {
  description = "ARN of the Lambda function"
  value       = aws_lambda_function.durable_function.arn
}

output "alias_arn" {
  description = "ARN of the function alias (use this for invocations)"
  value       = aws_lambda_alias.prod.arn
}
```

Untuk menerapkan dengan Terraform

```
terraform init
terraform plan
terraform apply
```

> **Note**
>
> Dukungan Terraform untuk fungsi tahan lama Lambda memerlukan AWS penyedia versi 6.25.0 atau yang lebih baru. Perbarui versi penyedia Anda jika Anda menggunakan versi yang lebih lama.

Fungsi tahan lama dari gambar wadah OCI

Anda juga dapat membuat fungsi Durable berdasarkan gambar kontainer. Untuk petunjuk tentang cara membuat gambar kontainer, lihat Runtime yang didukung untuk fungsi tahan lama.

AWS CDK

Dengan AWS CDK, Anda dapat mendefinisikan infrastruktur menggunakan bahasa pemrograman. Contoh berikut menunjukkan cara membuat fungsi tahan lama menggunakan TypeScript dari gambar wadah.

```
import * as cdk from 'aws-cdk-lib';
import * as lambda from 'aws-cdk-lib/aws-lambda';
import * as iam from 'aws-cdk-lib/aws-iam';
```

```
import { Construct } from 'constructs';

export class DurableFunctionStack extends cdk.Stack {
  constructor(scope: Construct, id: string, props?: cdk.StackProps) {
    super(scope, id, props);

    // Create the durable function
    const durableFunction = new lambda.DockerImageFunction(this, 'DurableFunction', {
      code: lambda.DockerImageCode.fromImageAsset('./lambda', {
        platform: cdk.aws_ecr_assets.Platform.LINUX_AMD64,
      }),
      functionName: 'myDurableFunction',
      memorySize: 512,
      timeout: cdk.Duration.seconds(30),
      durableConfig: { executionTimeout: cdk.Duration.hours(1), retentionPeriod:
 cdk.Duration.days(30) },
    });

    // Create version and alias
    const version = durableFunction.currentVersion;
    const alias = new lambda.Alias(this, 'ProdAlias', {
      aliasName: 'prod',
      version: version,
    });

    // Output the alias ARN
    new cdk.CfnOutput(this, 'FunctionAliasArn', {
      value: alias.functionArn,
      description: 'Use this ARN to invoke the durable function',
    });
  }
}
```

Untuk menerapkan tumpukan CDK

```
cdk deploy
```

AWS Serverless Application Model

AWS SAM menyederhanakan CloudFormation template untuk aplikasi tanpa server. Template berikut menciptakan fungsi tahan lama dengan AWS SAM.

```
AWSTemplateFormatVersion: '2010-09-09'
```

```
Transform: AWS::Serverless-2016-10-31
Description: Lambda durable function with SAM

Resources:
  DurableFunction:
    Type: AWS::Serverless::Function
    Properties:
      FunctionName: myDurableFunction
      PackageType: Image
      ImageUri: ./src
      DurableConfig:
        ExecutionTimeout: 3600
        RetentionPeriodInDays: 7
      Policies:
        - arn:aws:iam::aws:policy/service-role/AWSLambdaBasicDurableExecutionRolePolicy
      AutoPublishAlias: prod
    Metadata:
      DockerTag: latest
      DockerContext: ./src
      Dockerfile: Dockerfile

Outputs:
  FunctionArn:
    Description: Durable function ARN
    Value: !GetAtt DurableFunction.Arn
  AliasArn:
    Description: Function alias ARN (use this for invocations)
    Value: !Ref DurableFunction.Alias
```

Untuk menerapkan template SAM

```
sam build
sam deploy --guided
```

Pola konfigurasi umum

Terlepas dari alat iAc mana yang Anda gunakan, ikuti pola ini untuk fungsi tahan lama:

Aktifkan eksekusi yang tahan lama

Atur `DurableConfig` properti pada fungsi Anda untuk mengaktifkan eksekusi yang tahan lama. Properti ini hanya tersedia saat membuat fungsi. Anda tidak dapat mengaktifkan eksekusi tahan lama pada fungsi yang ada.

Berikan izin pos pemeriksaan

Lampirkan kebijakan `AWSLambdaBasicDurableExecutionRolePolicy` terkelola ke peran eksekusi. Kebijakan ini mencakup persyaratan `lambda:CheckpointDurableExecution` dan `lambda:GetDurableExecutionState` izin.

Gunakan ARN yang memenuhi syarat

Buat versi atau alias untuk fungsi Anda. Fungsi tahan lama memerlukan ARN yang memenuhi syarat (dengan versi atau alias) untuk pemanggilan. Gunakan `AutoPublishAlias` dalam AWS SAM atau buat versi eksplisit di CloudFormation, AWS CDK, dan Terraform.

Ketergantungan paket

Sertakan SDK eksekusi tahan lama dalam paket penerapan Anda. Untuk Node.js, instal`@aws/durable-execution-sdk-js`. Untuk Python, instal`aws-durable-execution-sdk-python`.

Langkah selanjutnya

Setelah menerapkan fungsi tahan lama Anda:

- Uji fungsi Anda menggunakan ARN yang memenuhi syarat (versi atau alias)
- Memantau kemajuan eksekusi di konsol Lambda di bawah tab Eksekusi tahan lama
- Melihat operasi pos pemeriksaan dalam peristiwa AWS CloudTrail data
- Tinjau CloudWatch Log untuk keluaran fungsi dan perilaku pemutaran ulang

Untuk informasi selengkapnya tentang penerapan fungsi Lambda dengan alat iAc, lihat:

- CloudFormation AWS::Lambda: :Referensi fungsi
- AWS CDK Dokumentasi modul lambda
- AWS SAM Panduan Pengembang

Konfigurasikan fungsi Lambda yang tahan lama

Pengaturan eksekusi yang tahan lama mengontrol berapa lama eksekusi yang tahan lama dapat berjalan di semua pemanggilannya, berapa lama AWS Lambda mempertahankan riwayat eksekusi, dan AWS KMS kunci mana yang mengenkripsi data eksekusi Anda. Konfigurasikan pengaturan ini untuk mengaktifkan eksekusi tahan lama untuk fungsi Anda.

Aktifkan eksekusi yang tahan lama

Konfigurasikan `DurableConfig` objek saat membuat fungsi Anda untuk mengatur batas waktu eksekusi dan retensi riwayat. Anda hanya dapat mengaktifkan eksekusi yang tahan lama saat membuat fungsi. Anda tidak dapat mengaktifkannya pada fungsi yang ada.

AWS CLI

```
aws lambda create-function \
  --function-name my-durable-function \
  --runtime nodejs24.x \
  --role arn:aws:iam::123456789012:role/my-durable-role \
  --handler index.handler \
  --zip-file fileb://function.zip \
  --durable-config '{"ExecutionTimeout": 3600, "RetentionPeriodInDays": 30,
 "KMSKeyArn": "arn:aws:kms:us-east-1:111122223333:key/key-id"}'
```

CloudFormation

```
Resources:
  MyDurableFunction:
    Type: AWS::Lambda::Function
    Properties:
      FunctionName: my-durable-function
      Runtime: nodejs24.x
      Handler: index.handler
      Code:
        ZipFile: |
          // Your durable function code
      DurableConfig:
        ExecutionTimeout: 3600
        RetentionPeriodInDays: 30
        KMSKeyArn: "arn:aws:kms:us-east-1:111122223333:key/key-id"
```

Parameter konfigurasi:

- `ExecutionTimeout` – Wajib. Waktu maksimum dalam hitungan detik eksekusi yang tahan lama dapat dijalankan sebelum Lambda menghentikan eksekusi. Batas waktu ini berlaku untuk seluruh eksekusi tahan lama, bukan pemanggilan fungsi individual. Anda harus menentukan `ExecutionTimeout` kapan pun Anda menyediakan`DurableConfig`; membuat fungsi dengan konfigurasi yang tahan lama tetapi tidak ada batas waktu eksekusi yang gagal dengan file. `InvalidParameterValueException` Kisaran yang valid: 1-31622400.
- `RetentionPeriodInDays` – Opsional. Jumlah hari untuk mempertahankan riwayat eksekusi setelah eksekusi yang tahan lama selesai. Setelah periode ini, riwayat eksekusi tidak lagi tersedia melalui `GetDurableExecutionHistory` API. Rentang yang valid: 1—90. Default: 14.
- `KMSKeyArn` – Opsional. ARN dari kunci yang dikelola pelanggan untuk mengenkripsi data eksekusi yang tahan lama. Untuk informasi selengkapnya, lihat Mengenkripsi data eksekusi tahan lama Lambda.

Untuk referensi API selengkapnya, lihat DurableConfigdi Referensi API Lambda.

Praktik terbaik konfigurasi

Ikuti praktik terbaik ini saat mengonfigurasi fungsi tahan lama untuk penggunaan produksi:

- Tetapkan batas waktu eksekusi yang sesuai - Konfigurasikan `ExecutionTimeout` berdasarkan durasi maksimum yang diharapkan alur kerja Anda. Jangan menetapkan batas waktu lama yang tidak perlu karena mempengaruhi biaya dan alokasi sumber daya.
- Retensi saldo dengan biaya penyimpanan - Tetapkan `RetentionPeriodInDays` berdasarkan persyaratan debugging dan audit Anda. Periode retensi yang lebih lama meningkatkan biaya penyimpanan.
- Ukuran status monitor — Objek status besar meningkatkan biaya penyimpanan dan dapat memengaruhi kinerja. Jaga status minimal dan gunakan penyimpanan eksternal untuk data besar.
- Konfigurasikan logging yang sesuai — Aktifkan pencatatan terperinci untuk memecahkan masalah alur kerja yang berjalan lama, tetapi pertimbangkan dampaknya pada volume dan biaya log.
- Enkripsi dengan AWS KMS kunci Anda sendiri — Setel `KMSKeyArn` ke kunci yang dikelola pelanggan untuk mengontrol rotasi kunci, akses audit, dan memenuhi persyaratan kepatuhan. Lihat Mengenkripsi AWS Lambda data eksekusi tahan lama.

Contoh konfigurasi produksi:

```
{
  "ExecutionTimeout": 86400,
  "RetentionPeriodInDays": 7,
  "KMSKeyArn": "arn:aws:kms:us-east-1:111122223333:key/key-id"
}
```

Contoh ini menetapkan batas waktu eksekusi 24 jam (86.400 detik), periode retensi 7 hari, dan kunci yang dikelola pelanggan untuk enkripsi.

Fungsi tahan lama atau Step Functions

Kedua fungsi Lambda tahan lama dan AWS Step Functions memungkinkan orkestrasi alur kerja yang andal dengan manajemen status otomatis dan pemulihan kegagalan. Mereka melayani preferensi pengembang yang berbeda dan pola arsitektur. Fungsi tahan lama dioptimalkan untuk pengembangan aplikasi dalam Lambda, sementara Step Functions dibangun untuk orkestrasi alur kerja di seluruh layanan. AWS

Kapan menggunakan fungsi yang tahan lama

Gunakan fungsi tahan lama saat:

- Tim Anda lebih menyukai bahasa pemrograman standar dan alat pengembangan yang sudah dikenal
- Logika aplikasi Anda terutama dalam fungsi Lambda
- Anda ingin kontrol halus atas status eksekusi dalam kode
- Anda sedang membangun aplikasi Lambda-sentris dengan kopling ketat antara alur kerja dan logika bisnis
- Anda ingin mengulangi dengan cepat tanpa beralih antara kode dan desainer visual/JSON

Kapan menggunakan Step Functions

Gunakan Step Functions saat:

- Anda memerlukan representasi alur kerja visual untuk visibilitas lintas tim
- Anda mengatur beberapa AWS layanan dan menginginkan integrasi asli tanpa kode SDK khusus
- Anda memerlukan infrastruktur tanpa pemeliharaan (tidak ada penambalan, pembaruan runtime)
- Pemangku kepentingan non-teknis perlu memahami dan memvalidasi logika alur kerja

Kerangka keputusan

Gunakan pertanyaan berikut untuk menentukan layanan mana yang sesuai dengan kasus penggunaan Anda:

- Apa fokus utama Anda? Pengembangan aplikasi di Lambda → fungsi tahan lama. Orkestrasi alur kerja di seluruh → Step AWS Functions.
- Apa model pemrograman pilihan Anda? Bahasa pemrograman standar → fungsi tahan lama. DSL berbasis grafik atau desainer visual → Step Functions.
- Berapa banyak AWS layanan yang terlibat? Terutama Lambda → fungsi tahan lama. Beberapa AWS layanan → Step Functions.
- Alat pengembangan apa yang Anda gunakan? Pengalaman pengembang Lambda, IDE dengan agen LLM, kerangka kerja pengujian unit khusus bahasa pemrograman,, AWS SAM, AWS CDK Toolkit → fungsi tahan lama. AWS Pembuat alur kerja visual, AWS CDK untuk memodelkan alur kerja → Step Functions.
- Siapa yang mengelola infrastruktur? Ingin fleksibilitas dalam Lambda → fungsi tahan lama. Ingin dikelola sepenuhnya, tanpa pemeliharaan → Step Functions.

Perbandingan fitur

Tabel berikut membandingkan fitur utama antara Step Functions dan Lambda fungsi tahan lama:

Fitur	AWS Step Functions	Lambda fungsi tahan lama
Fokus utama	Orkestrasi alur kerja lintas AWS	Pengembangan aplikasi di Lambda
Jenis Layanan	Layanan alur kerja mandiri dan berdedikasi	Berjalan di dalam Lambda
Model pemrograman	Berbasis grafik, DSL Bahasa Negara Amazon atau AWS CDK	Bahasa pemrograman standar (JavaScript/TypeScript, Python)
Alat pengembangan	Pembangun visual di ekstensi IDE AWS Konsol/Toolkit, AWS CDK	Lambda DX dalam agen IDE dan LLM, kerangka pengujian unit, ekstensi IDE Toolkit AWS SAM AWS

Fitur	AWS Step Functions	Lambda fungsi tahan lama
Integrasi	220+ AWS layanan, 16k APIs	Ekstensi model pemrogram an berbasis peristiwa Lambda (sumber acara)
Manajemen	Dikelola sepenuhnya, agnostik runtime, nol pemeliharaan (tidak ada penambalan, pembaruan runtime)	Dikelola dalam lingkungan Lambda
Terbaik untuk	Proses bisnis dan otomatisasi TI, pemrosesan data, alur kerja AI	Transaksi terdistribusi, logika aplikasi stateful, orkestrasi fungsi, pemrosesan data, alur kerja AI

Arsitektur hibrida

Banyak aplikasi mendapat manfaat dari menggunakan kedua layanan tersebut. Pola umum menggunakan fungsi tahan lama untuk logika tingkat aplikasi dalam Lambda, sementara Step Functions mengoordinasikan alur kerja tingkat tinggi di beberapa layanan di luar fungsi Lambda. AWS

Pertimbangan migrasi

Memulai kompleks yang sederhana dan berkembang: Mulailah dengan fungsi tahan lama untuk alur kerja yang berpusat pada Lambda. Tambahkan Step Functions saat Anda membutuhkan orkestrasi multi-layanan atau desain alur kerja visual.

Pengguna Step Functions yang ada: Keep Step Functions untuk alur kerja lintas layanan yang telah ditetapkan. Pertimbangkan fungsi tahan lama untuk logika aplikasi Lambda baru yang membutuhkan keandalan.

Sumber daya terkait

- Lambda fungsi tahan lama
- Mengatur fungsi Lambda dengan Step Functions
- Memulai dengan fungsi yang tahan lama

Contoh dan kasus penggunaan

Fungsi Lambda yang tahan lama memungkinkan Anda membangun aplikasi multi-langkah yang toleran terhadap kesalahan menggunakan operasi yang tahan lama seperti langkah dan menunggu. Dengan checkpointing otomatis dan model checkpoint-replay, di mana eksekusi dimulai ulang dari awal setelah kegagalan tetapi melewati pos pemeriksaan yang telah selesai, fungsi Anda dapat pulih dari kegagalan dan melanjutkan eksekusi tanpa kehilangan kemajuan.

> **Note**
>
> Untuk referensi SDK API, tutorial mulai cepat, panduan khusus bahasa, dan pola alur kerja tambahan, lihat Panduan Pengembang SDK Eksekusi AWS Tahan Lama.

Short-lived proses toleran kesalahan

Gunakan fungsi yang tahan lama untuk membangun operasi andal yang biasanya selesai dalam hitungan menit. Meskipun proses ini lebih pendek dari alur kerja yang berjalan lama, mereka masih mendapat manfaat dari pemeriksaan otomatis dan toleransi kesalahan di seluruh sistem terdistribusi. Fungsi tahan lama membantu memastikan proses multi-langkah Anda berhasil diselesaikan bahkan ketika panggilan layanan individu gagal, tanpa memerlukan penanganan kesalahan yang rumit atau kode manajemen status.

Skenario umum termasuk sistem pemesanan hotel, platform reservasi restoran, permintaan perjalanan berbagi perjalanan, pembelian tiket acara, dan peningkatan langganan SaaS. Skenario ini memiliki karakteristik umum: beberapa panggilan layanan yang harus diselesaikan bersama, kebutuhan untuk mencoba ulang otomatis pada kegagalan sementara, dan persyaratan untuk mempertahankan status konsisten di seluruh sistem terdistribusi.

Transaksi terdistribusi di seluruh layanan mikro

Mengkoordinasikan pembayaran, inventaris, dan pengiriman di beberapa layanan dengan rollback otomatis pada kegagalan. Setiap operasi layanan dibungkus dalam satu langkah, memastikan transaksi dapat pulih dari titik mana pun jika layanan gagal.

TypeScript

```
import { DurableContext, withDurableExecution } from "@aws/durable-execution-sdk-
js";
```

```
export const handler = withDurableExecution(
  async (event: any, context: DurableContext) => {
    const { orderId, amount, items } = event;

    // Reserve inventory across multiple warehouses
    const inventory = await context.step("reserve-inventory", async () => {
      return await inventoryService.reserve(items);
    });

    // Process payment
    const payment = await context.step("process-payment", async () => {
      return await paymentService.charge(amount);
    });

    // Create shipment
    const shipment = await context.step("create-shipment", async () => {
      return await shippingService.createShipment(orderId, inventory);
    });

    return { orderId, status: 'completed', shipment };
  }
);
```

Python

```
from aws_durable_execution_sdk_python import DurableContext, durable_execution

@durable_execution
def lambda_handler(event, context: DurableContext):
    order_id = event['orderId']
    amount = event['amount']
    items = event['items']

    # Reserve inventory across multiple warehouses
    inventory = context.step(
        lambda _: inventory_service.reserve(items),
        name='reserve-inventory'
    )

    # Process payment
```

```
    payment = context.step(
        lambda _: payment_service.charge(amount),
        name='process-payment'
    )

    # Create shipment
    shipment = context.step(
        lambda _: shipping_service.create_shipment(order_id, inventory),
        name='create-shipment'
    )

    return {'orderId': order_id, 'status': 'completed', 'shipment': shipment}
```

Jika ada langkah yang gagal, fungsi secara otomatis mencoba ulang dari pos pemeriksaan terakhir yang berhasil. Reservasi inventaris tetap ada meskipun pemrosesan pembayaran gagal sementara. Ketika fungsi mencoba lagi, ia melewatkan langkah inventaris yang telah selesai dan langsung melanjutkan ke pemrosesan pembayaran. Ini menghilangkan reservasi duplikat dan memastikan status konsisten di seluruh sistem terdistribusi Anda.

Pemrosesan pesanan dengan beberapa langkah

Memproses pesanan melalui validasi, otorisasi pembayaran, alokasi inventaris, dan pemenuhan dengan percobaan ulang dan pemulihan otomatis. Setiap langkah diperiksa, memastikan pesanan berlanjut bahkan jika langkah individu gagal dan coba lagi.

TypeScript

```
import { DurableContext, withDurableExecution } from "@aws/durable-execution-sdk-
js";

export const handler = withDurableExecution(
  async (event: any, context: DurableContext) => {
    const { orderId, customerId, items } = event;

    // Validate order details
    const validation = await context.step("validate-order", async () => {
      const customer = await customerService.validate(customerId);
      const itemsValid = await inventoryService.validateItems(items);
      return { customer, itemsValid };
    });
```

```
    if (!validation.itemsValid) {
      return { orderId, status: 'rejected', reason: 'invalid_items' };
    }

    // Authorize payment
    const authorization = await context.step("authorize-payment", async () => {
      return await paymentService.authorize(
        validation.customer.paymentMethod,
        calculateTotal(items)
      );
    });

    // Allocate inventory
    const allocation = await context.step("allocate-inventory", async () => {
      return await inventoryService.allocate(items);
    });

    // Fulfill order
    const fulfillment = await context.step("fulfill-order", async () => {
      return await fulfillmentService.createShipment({
        orderId,
        items: allocation.allocatedItems,
        address: validation.customer.shippingAddress
      });
    });

    return {
      orderId,
      status: 'completed',
      trackingNumber: fulfillment.trackingNumber
    };
  }
);
```

Python

```
from aws_durable_execution_sdk_python import DurableContext, durable_execution

@durable_execution
def lambda_handler(event, context: DurableContext):
```

```
    order_id = event['orderId']
    customer_id = event['customerId']
    items = event['items']

    # Validate order details
    def validate_order(_):
        customer = customer_service.validate(customer_id)
        items_valid = inventory_service.validate_items(items)
        return {'customer': customer, 'itemsValid': items_valid}

    validation = context.step(validate_order, name='validate-order')

    if not validation['itemsValid']:
        return {'orderId': order_id, 'status': 'rejected', 'reason':
'invalid_items'}

    # Authorize payment
    authorization = context.step(
        lambda _: payment_service.authorize(
            validation['customer']['paymentMethod'],
            calculate_total(items)
        ),
        name='authorize-payment'
    )

    # Allocate inventory
    allocation = context.step(
        lambda _: inventory_service.allocate(items),
        name='allocate-inventory'
    )

    # Fulfill order
    fulfillment = context.step(
        lambda _: fulfillment_service.create_shipment({
            'orderId': order_id,
            'items': allocation['allocatedItems'],
            'address': validation['customer']['shippingAddress']
        }),
        name='fulfill-order'
    )

    return {
        'orderId': order_id,
        'status': 'completed',
```

```
        'trackingNumber': fulfillment['trackingNumber']
    }
```

Pola ini memastikan pesanan tidak pernah terjebak dalam keadaan menengah. Jika validasi gagal, pesanan ditolak sebelum otorisasi pembayaran. Jika otorisasi pembayaran gagal, inventaris tidak dialokasikan. Setiap langkah dibangun di atas yang sebelumnya dengan percobaan ulang dan pemulihan otomatis.

Catatan

Pemeriksaan bersyarat `if (!validation.itemsValid)` berada di luar langkah dan dijalankan ulang selama pemutaran ulang. Ini aman karena deterministik—selalu menghasilkan hasil yang sama dengan objek validasi yang sama.

Long-running proses

Gunakan fungsi tahan lama untuk proses yang berlangsung berjam-jam, berhari-hari, atau berminggu-minggu. Operasi tunggu menangguhkan eksekusi tanpa menimbulkan biaya komputasi, membuat proses yang berjalan lama menjadi hemat biaya. Selama periode tunggu, fungsi Anda berhenti berjalan dan Lambda mendaur ulang lingkungan eksekusi. Ketika tiba waktunya untuk melanjutkan, Lambda memanggil fungsi Anda lagi dan memutar ulang dari pos pemeriksaan terakhir.

Model eksekusi ini membuat fungsi tahan lama ideal untuk proses yang perlu dijeda untuk waktu yang lama, baik menunggu keputusan manusia, respons sistem eksternal, jendela pemrosesan terjadwal, atau penundaan berbasis waktu. Anda hanya membayar untuk waktu komputasi aktif, bukan untuk menunggu.

Skenario umum termasuk proses persetujuan dokumen, pemrosesan batch terjadwal, proses orientasi multi-hari, proses uji coba berlangganan, dan sistem notifikasi tertunda. Skenario ini memiliki karakteristik umum: periode tunggu yang diperpanjang yang diukur dalam jam atau hari, kebutuhan untuk mempertahankan status eksekusi di seluruh penantian tersebut, dan persyaratan yang sensitif terhadap biaya di mana membayar waktu komputasi idle sangat mahal.

Human-in-the-loop persetujuan

Jeda eksekusi untuk tinjauan dokumen, persetujuan, atau keputusan sambil mempertahankan status eksekusi. Fungsi menunggu callback eksternal tanpa menghabiskan sumber daya, dilanjutkan secara otomatis ketika persetujuan diterima.

Pola ini penting untuk proses yang membutuhkan penilaian manusia atau validasi eksternal. Fungsi ditangguhkan pada titik panggilan balik, tidak menimbulkan biaya komputasi saat menunggu. Ketika seseorang mengirimkan keputusan mereka melalui API, Lambda memanggil fungsi Anda lagi dan memutar ulang dari pos pemeriksaan, melanjutkan dengan hasil persetujuan.

TypeScript

```
import { DurableContext, withDurableExecution } from "@aws/durable-execution-sdk-
js";

export const handler = withDurableExecution(
  async (event: any, context: DurableContext) => {
    const { documentId, reviewers } = event;

    // Step 1: Prepare document for review
    const prepared = await context.step("prepare-document", async () => {
      return await documentService.prepare(documentId);
    });

    // Step 2: Request approval with callback
    const approval = await context.waitForCallback(
      "approval-callback",
      async (callbackId) => {
        await notificationService.sendApprovalRequest({
          documentId,
          reviewers,
          callbackId,
          expiresIn: 86400
        });
      },
      {
        timeout: { seconds: 86400 }
      }
    );

    // Function resumes here when approval is received
    if (approval?.approved) {
      const finalized = await context.step("finalize-document", async () => {
        return await documentService.finalize(documentId, approval.comments);
      });

      return {
```

```
        status: 'approved',
        documentId,
        finalizedAt: finalized.timestamp
      };
    }

    // Handle rejection
    await context.step("archive-rejected", async () => {
      await documentService.archive(documentId, approval?.reason);
    });

    return {
      status: 'rejected',
      documentId,
      reason: approval?.reason
    };
  }
);
```

Python

```
from aws_durable_execution_sdk_python import DurableContext, durable_execution,
 WaitConfig

@durable_execution
def lambda_handler(event, context: DurableContext):
    document_id = event['documentId']
    reviewers = event['reviewers']

    # Step 1: Prepare document for review
    prepared = context.step(
        lambda _: document_service.prepare(document_id),
        name='prepare-document'
    )

    # Step 2: Request approval with callback
    def send_approval_request(callback_id):
        notification_service.send_approval_request({
            'documentId': document_id,
            'reviewers': reviewers,
            'callbackId': callback_id,
```

```
            'expiresIn': 86400
        })

    approval = context.wait_for_callback(
        send_approval_request,
        name='approval-callback',
        config=WaitConfig(timeout=86400)
    )

    # Function resumes here when approval is received
    if approval and approval.get('approved'):
        finalized = context.step(
            lambda _: document_service.finalize(document_id,
approval.get('comments')),
            name='finalize-document'
        )

        return {
            'status': 'approved',
            'documentId': document_id,
            'finalizedAt': finalized['timestamp']
        }

    # Handle rejection
    context.step(
        lambda _: document_service.archive(document_id, approval.get('reason') if
approval else None),
        name='archive-rejected'
    )

    return {
        'status': 'rejected',
        'documentId': document_id,
        'reason': approval.get('reason') if approval else None
    }
```

Ketika callback diterima dan fungsi Anda dilanjutkan, itu akan diputar ulang dari awal. Langkah persiapan-dokumen mengembalikan hasil checkpoint-nya secara instan. ForCallback Operasi tunggu juga kembali secara instan dengan hasil persetujuan yang disimpan alih-alih menunggu lagi. Eksekusi kemudian dilanjutkan ke langkah finalisasi atau arsip.

Multi-stage jaringan pipa data

Memproses kumpulan data besar melalui fase ekstraksi, transformasi, dan pemuatan dengan pos pemeriksaan antar tahapan. Setiap tahap dapat memakan waktu berjam-jam untuk diselesaikan, dan pos pemeriksaan memungkinkan pipa untuk melanjutkan dari tahap mana pun jika terputus.

Pola ini sangat ideal untuk alur kerja ETL, migrasi data, atau pekerjaan pemrosesan batch di mana Anda perlu memproses data secara bertahap dengan titik pemulihan di antaranya. Jika tahap gagal, pipa dilanjutkan dari tahap selesai terakhir daripada memulai kembali dari awal. Anda juga dapat menggunakan operasi tunggu untuk berhenti sejenak di antara tahapan; menghormati batas tarif, menunggu sistem hilir siap, atau menjadwalkan pemrosesan selama jam-jam sibuk.

TypeScript

```
import { DurableContext, withDurableExecution } from "@aws/durable-execution-sdk-
js";

export const handler = withDurableExecution(
  async (event: any, context: DurableContext) => {
    const { datasetId, batchSize } = event;

    // Stage 1: Extract data from source
    const extracted = await context.step("extract-data", async () => {
      const records = await sourceDatabase.extractRecords(datasetId);
      return { recordCount: records.length, records };
    });

    // Wait 5 minutes to respect source system rate limits
    await context.wait({ seconds: 300 });

    // Stage 2: Transform data in batches
    const transformed = await context.step("transform-data", async () => {
      const batches = chunkArray(extracted.records, batchSize);
      const results = [];

      for (const batch of batches) {
        const transformed = await transformService.processBatch(batch);
        results.push(transformed);
      }

      return { batchCount: batches.length, results };
    });
```

```
    // Wait until off-peak hours (e.g., 2 AM)
    const now = new Date();
    const targetHour = 2;
    const msUntilTarget = calculateMsUntilHour(now, targetHour);
    await context.wait({ seconds: Math.floor(msUntilTarget / 1000) });

    // Stage 3: Load data to destination
    const loaded = await context.step("load-data", async () => {
      let loadedCount = 0;

      for (const result of transformed.results) {
        await destinationDatabase.loadBatch(result);
        loadedCount += result.length;
      }

      return { loadedCount };
    });

    // Stage 4: Verify and finalize
    const verified = await context.step("verify-pipeline", async () => {
      const verification = await destinationDatabase.verifyRecords(datasetId);
      await pipelineService.markComplete(datasetId, verification);
      return verification;
    });

    return {
      datasetId,
      recordsProcessed: extracted.recordCount,
      batchesProcessed: transformed.batchCount,
      recordsLoaded: loaded.loadedCount,
      verified: verified.success
    };
  }
);
```

Python

```
from aws_durable_execution_sdk_python import DurableContext, durable_execution
from datetime import datetime
```

```
@durable_execution
def lambda_handler(event, context: DurableContext):
    dataset_id = event['datasetId']
    batch_size = event['batchSize']

    # Stage 1: Extract data from source
    def extract_data(_):
        records = source_database.extract_records(dataset_id)
        return {'recordCount': len(records), 'records': records}

    extracted = context.step(extract_data, name='extract-data')

    # Wait 5 minutes to respect source system rate limits
    context.wait(Duration.from_seconds(300))

    # Stage 2: Transform data in batches
    def transform_data(_):
        batches = chunk_array(extracted['records'], batch_size)
        results = []

        for batch in batches:
            transformed = transform_service.process_batch(batch)
            results.append(transformed)

        return {'batchCount': len(batches), 'results': results}

    transformed = context.step(transform_data, name='transform-data')

    # Wait until off-peak hours (e.g., 2 AM)
    now = datetime.now()
    target_hour = 2
    ms_until_target = calculate_ms_until_hour(now, target_hour)
    context.wait(ms_until_target // 1000)

    # Stage 3: Load data to destination
    def load_data(_):
        loaded_count = 0

        for result in transformed['results']:
            destination_database.load_batch(result)
            loaded_count += len(result)

        return {'loadedCount': loaded_count}
```

```
    loaded = context.step(load_data, name='load-data')

    # Stage 4: Verify and finalize
    def verify_pipeline(_):
        verification = destination_database.verify_records(dataset_id)
        pipeline_service.mark_complete(dataset_id, verification)
        return verification

    verified = context.step(verify_pipeline, name='verify-pipeline')

    return {
        'datasetId': dataset_id,
        'recordsProcessed': extracted['recordCount'],
        'batchesProcessed': transformed['batchCount'],
        'recordsLoaded': loaded['loadedCount'],
        'verified': verified['success']
    }
```

Setiap tahap dibungkus dalam satu langkah, menciptakan pos pemeriksaan yang memungkinkan pipa untuk melanjutkan dari tahap apa pun jika terputus. Penantian 5 menit antara ekstrak dan transformasi menghormati batas laju sistem sumber tanpa mengkonsumsi sumber daya komputasi, sementara menunggu hingga jam 2 pagi menjadwalkan operasi beban yang mahal selama jam-jam di luar sibuk.

Catatan

`new Date()`Panggilan dan `calculateMsUntilHour()` fungsi berada di luar langkah dan dijalankan kembali selama pemutaran ulang. Untuk operasi berbasis waktu yang harus konsisten di seluruh tayangan ulang, hitung stempel waktu di dalam langkah atau gunakan hanya untuk durasi tunggu (yang diperiksa).

Pola lanjutan

Gunakan fungsi tahan lama untuk membangun aplikasi multi-langkah kompleks yang menggabungkan beberapa operasi tahan lama, eksekusi paralel, pemrosesan array, logika bersyarat, dan polling. Pola ini memungkinkan Anda membangun aplikasi canggih yang mengoordinasikan banyak tugas sambil mempertahankan toleransi kesalahan dan pemulihan otomatis.

Pola lanjutan melampaui langkah-langkah berurutan sederhana. Anda dapat menjalankan operasi secara bersamaan dengan`parallel()`, memproses array dengan`map()`, menunggu kondisi

eksternal dengan`waitForCondition()`, dan menggabungkan primitif ini untuk membangun aplikasi yang andal. Setiap operasi yang tahan lama membuat pos pemeriksaannya sendiri, sehingga aplikasi Anda dapat pulih dari titik mana pun jika terputus.

Proses orientasi pengguna

Memandu pengguna melalui pendaftaran, verifikasi email, pengaturan profil, dan konfigurasi awal dengan penanganan coba lagi. Contoh ini menggabungkan langkah-langkah berurutan, callback, dan logika bersyarat untuk membuat proses orientasi yang lengkap.

TypeScript

```
import { DurableContext, withDurableExecution } from "@aws/durable-execution-sdk-
js";

export const handler = withDurableExecution(
  async (event: any, context: DurableContext) => {
    const { userId, email } = event;

    // Step 1: Create user account
    const user = await context.step("create-account", async () => {
      return await userService.createAccount(userId, email);
    });

    // Step 2: Send verification email
    await context.step("send-verification", async () => {
      return await emailService.sendVerification(email);
    });

    // Step 3: Wait for email verification (up to 48 hours)
    const verified = await context.waitForCallback(
      "email-verification",
      async (callbackId) => {
        await notificationService.sendVerificationLink({
          email,
          callbackId,
          expiresIn: 172800
        });
      },
      {
        timeout: { seconds: 172800 }
      }
```

```
  );

  if (!verified) {
    await context.step("send-reminder", async () => {
      await emailService.sendReminder(email);
    });

    return {
      status: "verification_timeout",
      userId,
      message: "Email verification not completed within 48 hours"
    };
  }

  // Step 4: Initialize user profile in parallel
  const setupResults = await context.parallel("profile-setup", [
    async (ctx: DurableContext) => {
      return await ctx.step("create-preferences", async () => {
        return await preferencesService.createDefaults(userId);
      });
    },

    async (ctx: DurableContext) => {
      return await ctx.step("setup-notifications", async () => {
        return await notificationService.setupDefaults(userId);
      });
    },

    async (ctx: DurableContext) => {
      return await ctx.step("create-welcome-content", async () => {
        return await contentService.createWelcome(userId);
      });
    }
  ]);

  // Step 5: Send welcome email
  await context.step("send-welcome", async () => {
    const [preferences, notifications, content] = setupResults.getResults();
    return await emailService.sendWelcome({
      email,
      preferences,
      notifications,
      content
    });
```

```
    });

    return {
      status: "onboarding_complete",
      userId,
      completedAt: new Date().toISOString()
    };
  }
);
```

Python

```
from aws_durable_execution_sdk_python import DurableContext, durable_execution,
 WaitConfig
from datetime import datetime

@durable_execution
def lambda_handler(event, context: DurableContext):
    user_id = event['userId']
    email = event['email']

    # Step 1: Create user account
    user = context.step(
        lambda _: user_service.create_account(user_id, email),
        name='create-account'
    )

    # Step 2: Send verification email
    context.step(
        lambda _: email_service.send_verification(email),
        name='send-verification'
    )

    # Step 3: Wait for email verification (up to 48 hours)
    def send_verification_link(callback_id):
        notification_service.send_verification_link({
            'email': email,
            'callbackId': callback_id,
            'expiresIn': 172800
        })
```

```
    verified = context.wait_for_callback(
        send_verification_link,
        name='email-verification',
        config=WaitConfig(timeout=172800)
    )

    if not verified:
        context.step(
            lambda _: email_service.send_reminder(email),
            name='send-reminder'
        )

        return {
            'status': 'verification_timeout',
            'userId': user_id,
            'message': 'Email verification not completed within 48 hours'
        }

    # Step 4: Initialize user profile in parallel
    def create_preferences(ctx: DurableContext):
        return ctx.step(
            lambda _: preferences_service.create_defaults(user_id),
            name='create-preferences'
        )

    def setup_notifications(ctx: DurableContext):
        return ctx.step(
            lambda _: notification_service.setup_defaults(user_id),
            name='setup-notifications'
        )

    def create_welcome_content(ctx: DurableContext):
        return ctx.step(
            lambda _: content_service.create_welcome(user_id),
            name='create-welcome-content'
        )

    setup_results = context.parallel(
        [create_preferences, setup_notifications, create_welcome_content],
        name='profile-setup'
    )

    # Step 5: Send welcome email
    def send_welcome(_):
```

```
        results = setup_results.get_results()
        preferences, notifications, content = results[0], results[1], results[2]
        return email_service.send_welcome({
            'email': email,
            'preferences': preferences,
            'notifications': notifications,
            'content': content
        })

    context.step(send_welcome, name='send-welcome')

    return {
        'status': 'onboarding_complete',
        'userId': user_id,
        'completedAt': datetime.now().isoformat()
    }
```

Proses ini menggabungkan langkah-langkah berurutan dengan pos pemeriksaan untuk pembuatan akun dan pengiriman email, kemudian berhenti hingga 48 jam menunggu verifikasi email tanpa menghabiskan sumber daya. Logika bersyarat menangani jalur yang berbeda berdasarkan apakah verifikasi selesai atau habis waktu. Tugas penyiapan profil berjalan secara bersamaan menggunakan operasi paralel untuk mengurangi total waktu eksekusi, dan setiap langkah mencoba ulang secara otomatis pada kegagalan sementara untuk membantu memastikan orientasi selesai dengan andal.

Pemanggilan berantai di seluruh fungsi

Memanggil fungsi Lambda lainnya dari dalam fungsi tahan lama menggunakan. `context.invoke()` Fungsi pemanggilan ditangguhkan sambil menunggu fungsi yang dipanggil selesai, membuat pos pemeriksaan yang mempertahankan hasilnya. Jika fungsi pemanggilan terputus setelah fungsi yang dipanggil selesai, fungsi tersebut dilanjutkan dengan hasil yang disimpan tanpa memanggil kembali fungsi tersebut.

Gunakan pola ini ketika Anda memiliki fungsi khusus yang menangani domain tertentu (validasi pelanggan, pemrosesan pembayaran, manajemen inventaris) dan perlu mengoordinasikannya dalam alur kerja. Setiap fungsi mempertahankan logikanya sendiri dan dapat dipanggil oleh beberapa fungsi orkestrator, menghindari duplikasi kode.

TypeScript

```
import { DurableContext, withDurableExecution } from "@aws/durable-execution-sdk-
js";

// Main orchestrator function
export const handler = withDurableExecution(
  async (event: any, context: DurableContext) => {
    const { orderId, customerId } = event;

    // Step 1: Validate customer by invoking customer service function
    const customer = await context.invoke(
      "validate-customer",
      "arn:aws:lambda:us-east-1:123456789012:function:customer-service:1",
      { customerId }
    );

    if (!customer.isValid) {
      return { orderId, status: "rejected", reason: "invalid_customer" };
    }

    // Step 2: Check inventory by invoking inventory service function
    const inventory = await context.invoke(
      "check-inventory",
      "arn:aws:lambda:us-east-1:123456789012:function:inventory-service:1",
      { orderId, items: event.items }
    );

    if (!inventory.available) {
      return { orderId, status: "rejected", reason: "insufficient_inventory" };
    }

    // Step 3: Process payment by invoking payment service function
    const payment = await context.invoke(
      "process-payment",
      "arn:aws:lambda:us-east-1:123456789012:function:payment-service:1",
      {
        customerId,
        amount: inventory.totalAmount,
        paymentMethod: customer.paymentMethod
      }
    );
```

```
    // Step 4: Create shipment by invoking fulfillment service function
    const shipment = await context.invoke(
      "create-shipment",
      "arn:aws:lambda:us-east-1:123456789012:function:fulfillment-service:1",
      {
        orderId,
        items: inventory.allocatedItems,
        address: customer.shippingAddress
      }
    );

    return {
      orderId,
      status: "completed",
      trackingNumber: shipment.trackingNumber,
      estimatedDelivery: shipment.estimatedDelivery
    };
  }
);
```

Python

```
from aws_durable_execution_sdk_python import DurableContext, durable_execution

# Main orchestrator function
@durable_execution
def lambda_handler(event, context: DurableContext):
    order_id = event['orderId']
    customer_id = event['customerId']

    # Step 1: Validate customer by invoking customer service function
    customer = context.invoke(
        'arn:aws:lambda:us-east-1:123456789012:function:customer-service:1',
        {'customerId': customer_id},
        name='validate-customer'
    )

    if not customer['isValid']:
        return {'orderId': order_id, 'status': 'rejected', 'reason':
 'invalid_customer'}
```

```
    # Step 2: Check inventory by invoking inventory service function
    inventory = context.invoke(
        'arn:aws:lambda:us-east-1:123456789012:function:inventory-service:1',
        {'orderId': order_id, 'items': event['items']},
        name='check-inventory'
    )

    if not inventory['available']:
        return {'orderId': order_id, 'status': 'rejected', 'reason':
'insufficient_inventory'}

    # Step 3: Process payment by invoking payment service function
    payment = context.invoke(
        'arn:aws:lambda:us-east-1:123456789012:function:payment-service:1',
        {
            'customerId': customer_id,
            'amount': inventory['totalAmount'],
            'paymentMethod': customer['paymentMethod']
        },
        name='process-payment'
    )

    # Step 4: Create shipment by invoking fulfillment service function
    shipment = context.invoke(
        'arn:aws:lambda:us-east-1:123456789012:function:fulfillment-service:1',
        {
            'orderId': order_id,
            'items': inventory['allocatedItems'],
            'address': customer['shippingAddress']
        },
        name='create-shipment'
    )

    return {
        'orderId': order_id,
        'status': 'completed',
        'trackingNumber': shipment['trackingNumber'],
        'estimatedDelivery': shipment['estimatedDelivery']
    }
```

Setiap pemanggilan membuat pos pemeriksaan dalam fungsi orkestrator. Jika orkestrator terganggu setelah validasi pelanggan selesai, ia melanjutkan dari pos pemeriksaan itu dengan data pelanggan yang disimpan, melewatkan pemanggilan validasi. Ini mencegah panggilan duplikat ke layanan hilir dan memastikan eksekusi yang konsisten di seluruh interupsi.

Fungsi yang dipanggil dapat berupa fungsi Lambda apa pun, termasuk fungsi yang tahan lama dengan alur kerja multi-langkahnya sendiri, menunggu, dan pos pemeriksaan. Fungsi pemanggilan menunggu eksekusi yang tahan lama selesai dan kemudian menerima hasil akhir.

> **Note**
>
> Cross-account pemanggilan tidak didukung. Semua fungsi yang dipanggil harus berada di AWS akun yang sama dengan fungsi pemanggilan.

Pemrosesan batch dengan pos pemeriksaan

Memproses jutaan catatan dengan pemulihan otomatis dari pos pemeriksaan terakhir yang berhasil setelah kegagalan. Contoh ini menunjukkan bagaimana fungsi yang tahan lama menggabungkan `map()` operasi dengan chunking dan pembatasan laju untuk menangani pemrosesan data skala besar.

TypeScript

```
import { DurableContext, withDurableExecution } from "@aws/durable-execution-sdk-
js";

interface Batch {
  batchIndex: number;
  recordIds: string[];
}

export const handler = withDurableExecution(
  async (event: any, context: DurableContext) => {
    const { datasetId, batchSize = 1000 } = event;

    // Step 1: Get all record IDs to process
    const recordIds = await context.step("fetch-record-ids", async () => {
      return await dataService.getRecordIds(datasetId);
    });
```

```
    // Step 2: Split into batches
    const batches: Batch[] = [];
    for (let i = 0; i < recordIds.length; i += batchSize) {
      batches.push({
        batchIndex: Math.floor(i / batchSize),
        recordIds: recordIds.slice(i, i + batchSize)
      });
    }

    // Step 3: Process batches with controlled concurrency
    const batchResults = await context.map(
      "process-batches",
      batches,
      async (ctx: DurableContext, batch: Batch, index: number) => {
        const processed = await ctx.step(`batch-${batch.batchIndex}`, async () => {
          const results = [];
          for (const recordId of batch.recordIds) {
            const result = await recordService.process(recordId);
            results.push(result);
          }
          return results;
        });

        const validated = await ctx.step(`validate-${batch.batchIndex}`, async () =>
{
          return await validationService.validateBatch(processed);
        });

        return {
          batchIndex: batch.batchIndex,
          recordCount: batch.recordIds.length,
          successCount: validated.successCount,
          failureCount: validated.failureCount
        };
      },
      {
        maxConcurrency: 5
      }
    );

    // Step 4: Aggregate results
    const summary = await context.step("aggregate-results", async () => {
      const results = batchResults.getResults();
```

```
      const totalSuccess = results.reduce((sum, r) => sum + r.successCount, 0);
      const totalFailure = results.reduce((sum, r) => sum + r.failureCount, 0);

      return {
        datasetId,
        totalRecords: recordIds.length,
        batchesProcessed: batches.length,
        successCount: totalSuccess,
        failureCount: totalFailure,
        completedAt: new Date().toISOString()
      };
    });

    return summary;
  }
);
```

Python

```
from aws_durable_execution_sdk_python import DurableContext, durable_execution,
 MapConfig
from datetime import datetime
from typing import List, Dict

@durable_execution
def lambda_handler(event, context: DurableContext):
    dataset_id = event['datasetId']
    batch_size = event.get('batchSize', 1000)

    # Step 1: Get all record IDs to process
    record_ids = context.step(
        lambda _: data_service.get_record_ids(dataset_id),
        name='fetch-record-ids'
    )

    # Step 2: Split into batches
    batches = []
    for i in range(0, len(record_ids), batch_size):
        batches.append({
            'batchIndex': i // batch_size,
            'recordIds': record_ids[i:i + batch_size]
```

```
        })

    # Step 3: Process batches with controlled concurrency
    def process_batch(ctx: DurableContext, batch: Dict, index: int):
        batch_index = batch['batchIndex']

        def process_records(_):
            results = []
            for record_id in batch['recordIds']:
                result = record_service.process(record_id)
                results.append(result)
            return results

        processed = ctx.step(process_records, name=f'batch-{batch_index}')

        validated = ctx.step(
            lambda _: validation_service.validate_batch(processed),
            name=f'validate-{batch_index}'
        )

        return {
            'batchIndex': batch_index,
            'recordCount': len(batch['recordIds']),
            'successCount': validated['successCount'],
            'failureCount': validated['failureCount']
        }

    batch_results = context.map(
        process_batch,
        batches,
        name='process-batches',
        config=MapConfig(max_concurrency=5)
    )

    # Step 4: Aggregate results
    def aggregate_results(_):
        results = batch_results.get_results()
        total_success = sum(r['successCount'] for r in results)
        total_failure = sum(r['failureCount'] for r in results)

        return {
            'datasetId': dataset_id,
            'totalRecords': len(record_ids),
            'batchesProcessed': len(batches),
```

```
            'successCount': total_success,
            'failureCount': total_failure,
            'completedAt': datetime.now().isoformat()
        }

    summary = context.step(aggregate_results, name='aggregate-results')

    return summary
```

Catatan dibagi menjadi batch yang dapat dikelola untuk menghindari memori yang berlebihan atau layanan hilir, kemudian beberapa batch diproses secara bersamaan dengan mengendalikan paralelisme. `maxConcurrency` Setiap batch memiliki pos pemeriksaannya sendiri, jadi kegagalan hanya mencoba lagi batch yang gagal daripada memproses ulang semua catatan. Pola ini sangat ideal untuk pekerjaan ETL, migrasi data, atau operasi massal di mana pemrosesan dapat memakan waktu berjam-jam.

Langkah selanjutnya

- Jelajahi konsep dasar untuk memahami DurableContext, langkah, dan menunggu
- Tinjau praktik terbaik untuk menulis kode deterministik dan mengoptimalkan kinerja
- Pelajari tentang menguji fungsi tahan lama secara lokal dan di cloud
- Bandingkan fungsi tahan lama dengan Step Functions untuk memahami kapan setiap pendekatan paling efektif. Lihat Fungsi tahan lama atau Step Functions.

Keamanan dan izin untuk fungsi tahan lama Lambda

AWS Lambda fungsi tahan lama memerlukan izin IAM khusus untuk menjalankan dan mengelola eksekusi yang tahan lama. Ikuti prinsip hak istimewa paling sedikit dengan hanya memberikan izin yang dibutuhkan setiap kepala sekolah.

Izin IAM untuk fungsi tahan lama

Untuk tindakan API eksekusi yang paling tahan lama, Lambda mengotorisasi permintaan terhadap ARN eksekusi tahan lama, bukan fungsi ARN. Eksekusi tahan lama adalah sub-sumber daya dari versi fungsi, dan ARN-nya memperluas fungsi ARN:

`arn:aws:lambda:`*`region`*`:`*`account-id`*`:function:`*`function-name`*`:`*`qualifier`*`/durable-execution/`*`execution-name`*`/`*`execution-id`*

Kualifikasi dalam eksekusi tahan lama ARN selalu `$LATEST` atau nomor versi.

Jika kebijakan Anda menentukan fungsi ARN yang tidak memenuhi syarat dalam elemen Resource, Lambda menolak permintaan karena ARN yang tidak memenuhi syarat tidak pernah cocok dengan ARN eksekusi yang tahan lama. Untuk memberikan akses, tambahkan pola kualifikasi ke fungsi ARN:

- `arn:aws:lambda:us-east-1:123456789012:function:my-function:*`— Cocokkan eksekusi tahan lama dari setiap versi.
- `arn:aws:lambda:us-east-1:123456789012:function:my-function:1/durable-execution/*`— Hanya cocok dengan eksekusi tahan lama versi 1.

Sebagian besar kebijakan menggunakan `:*` formulir karena nomor versi dapat berubah dengan setiap penerapan.

> **Alias tidak berlaku untuk izin eksekusi yang tahan lama**
>
> Anda dapat memanggil fungsi tahan lama melalui alias, tetapi ARN eksekusi yang dihasilkan berisi nomor versi yang diselesaikan.

Izin peran eksekusi

Peran eksekusi fungsi tahan lama Anda harus memiliki izin untuk membuat pos pemeriksaan dan mengambil status eksekusi. Kebijakan berikut menunjukkan izin minimum yang diperlukan:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "lambda:CheckpointDurableExecution",
                "lambda:GetDurableExecutionState"
            ],
            "Resource": "arn:aws:lambda:region:account-id:function:function-name:*"
        }
    ]
```

```
}
```

Saat Anda membuat fungsi tahan lama menggunakan konsol, Lambda secara otomatis menambahkan izin ini ke peran eksekusi. Jika Anda membuat fungsi menggunakan AWS CLI atau AWS CloudFormation, tambahkan izin ini ke peran eksekusi Anda.

Agar Lambda dapat mengambil peran eksekusi Anda, kebijakan kepercayaan peran harus menentukan Lambda service principal (`lambda.amazonaws.com`) sebagai layanan tepercaya. Contoh berikut menunjukkan kebijakan kepercayaan yang memberikan izin Lambda untuk mengambil peran.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "lambda.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Prinsip hak istimewa terkecil

Lingkup `Resource` elemen ke ARN fungsi tertentu alih-alih menggunakan wildcard. Ini membatasi peran eksekusi untuk operasi pos pemeriksaan hanya untuk fungsi yang membutuhkannya.

Example Izin cakupan untuk beberapa fungsi

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "lambda:CheckpointDurableExecution",
                "lambda:GetDurableExecutionState"
            ],
            "Resource": [
```

```
                "arn:aws:lambda:us-east-1:123456789012:function:orderProcessor:*",
                "arn:aws:lambda:us-east-1:123456789012:function:paymentHandler:*"
            ]
        }
    ]
}
```

Atau, Anda dapat menggunakan kebijakan terkel AWS ola `AWSLambdaBasicDurableExecutionRolePolicy` yang mencakup izin eksekusi tahan lama yang diperlukan bersama dengan izin eksekusi Lambda dasar untuk Amazon CloudWatch Logs.

Izin untuk mengelola eksekusi yang tahan lama

Lambda mengotorisasi`lambda:GetDurableExecution`,`lambda:GetDurableExecutionHistory`, dan `lambda:StopDurableExecution` tindakan terhadap eksekusi tahan lama ARN.

Untuk membuat daftar eksekusi fungsi yang tahan lama, gunakan`lambda:ListDurableExecutionsByFunction`. Lambda mengotorisasi tindakan ini terhadap fungsi ARN dari permintaan. Aturan berikut berlaku untuk pencocokan ARN:

- Kebijakan yang memberikan fungsi yang tidak memenuhi syarat ARN hanya cocok dengan permintaan yang menggunakan nama fungsi yang tidak memenuhi syarat.
- Kebijakan yang memberikan ARN yang memenuhi syarat hanya akan mencocokkan permintaan yang menggunakan nama fungsi yang memenuhi syarat.

Untuk informasi selengkapnya, lihat the section called “Mereferensikan fungsi di bagian Sumber Daya kebijakan”.

Untuk mencocokkan permintaan yang dibuat dengan salah satu bentuk nama fungsi, sertakan fungsi ARN yang tidak memenuhi syarat dan fungsi yang memenuhi syarat, seperti yang ditunjukkan oleh kebijakan berikut:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ManageDurableExecutions",
            "Effect": "Allow",
            "Action": [
```

```
                "lambda:GetDurableExecution",
                "lambda:GetDurableExecutionHistory",
                "lambda:StopDurableExecution"
            ],
            "Resource": "arn:aws:lambda:us-
east-1:123456789012:function:myDurableFunction:*"
        },
        {
            "Sid": "ListDurableExecutions",
            "Effect": "Allow",
            "Action": "lambda:ListDurableExecutionsByFunction",
            "Resource": [
                "arn:aws:lambda:us-east-1:123456789012:function:myDurableFunction",
                "arn:aws:lambda:us-east-1:123456789012:function:myDurableFunction:*"
            ]
        }
    ]
}
```

Izin untuk mengirim panggilan balik

Sistem eksternal yang mengirimkan hasil callback menggunakan `lambda:SendDurableExecutionCallbackHeartbeat` tindakan`lambda:SendDurableExecutionCallbackSuccess`,`lambda:SendDurableExecutionCall` dan.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "SendCallbacks",
            "Effect": "Allow",
            "Action": [
                "lambda:SendDurableExecutionCallbackSuccess",
                "lambda:SendDurableExecutionCallbackFailure",
                "lambda:SendDurableExecutionCallbackHeartbeat"
            ],
            "Resource": "arn:aws:lambda:us-
east-1:123456789012:function:myDurableFunction:*"
        }
    ]
}
```

Lambda mengotorisasi permintaan panggilan balik terhadap ARN dari eksekusi tahan lama yang menciptakan callback.

Enkripsi negara

Fungsi tahan lama Lambda mengenkripsi data eksekusi saat diam. Setiap eksekusi fungsi mempertahankan status terisolasi yang tidak dapat diakses oleh eksekusi lain. Anda dapat mengonfigurasi kunci yang dikelola pelanggan untuk mengenkripsi data eksekusi yang tahan lama. Untuk informasi selengkapnya, lihat Mengenkripsi AWS Lambda data eksekusi tahan lama.

Data pos pemeriksaan meliputi:

- Hasil langkah dan kesalahan
- Input panggil yang dirantai

Untuk daftar lengkap data eksekusi tahan lama yang dienkripsi saat diam, lihatApa yang dienkripsi.

Semua data dienkripsi saat transit menggunakan TLS saat Lambda membaca atau menulis data pos pemeriksaan.

Enkripsi kustom dengan serializer dan deserializer khusus

Untuk persyaratan keamanan kritis, Anda dapat menerapkan mekanisme enkripsi dan dekripsi Anda sendiri menggunakan serializer dan deserializer khusus () menggunakan SDK tahan lama. SerDer Pendekatan ini memberi Anda kontrol penuh atas kunci enkripsi dan algoritma yang digunakan untuk melindungi data pos pemeriksaan.

> **Important**
>
> Saat Anda menggunakan enkripsi khusus, Anda kehilangan visibilitas hasil operasi di konsol Lambda dan respons API. Data pos pemeriksaan muncul dienkripsi dalam riwayat eksekusi dan tidak dapat diperiksa tanpa dekripsi.

Kebutuhan peran eksekusi fungsi Anda `kms:Encrypt` dan `kms:Decrypt` izin untuk AWS KMS kunci yang digunakan dalam SerDer implementasi kustom.

CloudTrail penebangan

Lambda mencatat operasi pos pemeriksaan sebagai peristiwa data di. AWS CloudTrail Anda dapat menggunakannya CloudTrail untuk mengaudit saat pos pemeriksaan dibuat, melacak perubahan status eksekusi, dan memantau akses ke data eksekusi yang tahan lama.

Operasi pos pemeriksaan muncul di CloudTrail log dengan nama acara berikut:

- `CheckpointDurableExecution`- Tercatat saat langkah selesai dan membuat pos pemeriksaan
- `GetDurableExecutionState`- Tercatat saat Lambda mengambil status eksekusi selama pemutaran ulang

Untuk mengaktifkan pencatatan peristiwa data untuk fungsi tahan lama, konfigurasikan CloudTrail jejak untuk mencatat peristiwa data Lambda. Untuk informasi selengkapnya, lihat Pencatatan peristiwa data di Panduan CloudTrail Pengguna.

Contoh: entri CloudTrail log untuk operasi pos pemeriksaan

```
{
    "eventVersion": "1.08",
    "eventTime": "2024-11-16T10:30:45Z",
    "eventName": "CheckpointDurableExecution",
    "eventSource": "lambda.amazonaws.com",
    "requestParameters": {
        "functionName": "myDurableFunction",
        "executionId": "exec-abc123",
        "stepId": "step-1"
    },
    "responseElements": null,
    "eventType": "AwsApiCall"
}
```

Cross-account pertimbangan

Jika Anda memanggil fungsi tahan lama di seluruh AWS akun, akun pemanggil memerlukan `lambda:InvokeFunction` izin, tetapi operasi pos pemeriksaan selalu menggunakan peran eksekusi di akun fungsi. Akun pemanggil tidak dapat mengakses data pos pemeriksaan atau status eksekusi secara langsung.

Isolasi ini memastikan bahwa data pos pemeriksaan tetap aman di dalam akun fungsi, bahkan ketika dipanggil dari akun eksternal.

Fitur keamanan Lambda yang diwariskan

Fungsi tahan lama mewarisi semua fitur keamanan, tata kelola, dan kepatuhan dari Lambda. Ini termasuk konektivitas VPC, enkripsi variabel lingkungan, antrian surat mati, konkurensi cadangan, URL fungsi, penandatanganan kode, dan sertifikasi kepatuhan seperti SOC, PCI DSS, dan HIPAA.

Untuk informasi terperinci tentang fitur keamanan Lambda, lihat Keamanan AWS Lambda di Panduan Pengembang Lambda. Satu-satunya pertimbangan keamanan tambahan untuk fungsi tahan lama adalah izin IAM yang didokumentasikan dalam panduan ini.

Mengenkripsi AWS Lambda data eksekusi tahan lama

Dengan fungsi yang AWS Lambda tahan lama, Anda dapat membangun aplikasi multi-langkah yang tangguh yang berjalan hingga satu tahun, menggunakan pos pemeriksaan untuk memulihkan setiap eksekusi dari kegagalan dengan memutar ulang pekerjaan yang telah selesai.

Lambda selalu mengenkripsi data eksekusi yang tahan lama saat istirahat. Anda juga dapat mengonfigurasi kunci terkelola AWS KMS pelanggan Anda sendiri pada fungsi untuk kontrol rotasi, visibilitas audit, atau kepatuhan. Dengan kunci yang dikelola pelanggan, hanya prinsipal yang Anda otorisasi yang dapat membaca data eksekusi.

Cara kerja enkripsi

Eksekusi Lambda yang tahan lama menggunakan kunci KMS yang sama dengan yang dimulai sepanjang masa pakainya. Mengubah atau menghapus tombol pada fungsi hanya memengaruhi eksekusi yang dimulai setelah perubahan. Lihat Ketika kunci yang dikelola pelanggan tidak tersedia bagaimana eksekusi tahan lama berperilaku ketika kuncinya tidak tersedia.

Kunci yang dikelola pelanggan dikenakan AWS KMS biaya standar. Untuk detail harganya, lihat Harga AWS Key Management Service.

Apa yang dienkripsi

Saat Anda mengonfigurasi kunci terkelola pelanggan pada fungsi tahan lama, Lambda menggunakan kunci tersebut untuk mengenkripsi data eksekusi tahan lama berikut saat istirahat:

- Muatan input yang Anda lewati dengan setiap `Invoke` permintaan.
- Data pos pemeriksaan disimpan oleh `CheckpointDurableExecution` API, termasuk hasil langkah, kesalahan langkah, dan input pemanggilan berantai.

- Hasil eksekusi dan kesalahan.
- Hasil callback dan kesalahan yang Anda kirimkan melalui `SendDurableExecutionCallbackSuccess` dan`SendDurableExecutionCallbackFailure`.

> **Function-level dan kunci eksekusi yang tahan lama bersifat independen**
>
> Tingkat fungsi `KMSKeyArn` yang mengenkripsi variabel lingkungan, paket penerapan .zip, dan SnapStartsnapshot terpisah dari yang mengenkripsi data eksekusi yang tahan lama. `KMSKeyArn` `DurableConfig` Pengaturan satu tidak mengatur yang lain. Anda dapat menggunakan tombol KMS yang sama untuk keduanya, atau Anda dapat menggunakan tombol KMS yang berbeda.

Siapkan enkripsi kunci yang dikelola pelanggan

Menyiapkan enkripsi kunci yang dikelola pelanggan untuk fungsi yang tahan lama adalah proses tiga langkah. Selesaikan langkah-langkah berikut secara berurutan.

Buat kunci terkelola pelanggan

Fungsi tahan lama mendukung kunci KMS enkripsi simetris di AWS Wilayah yang sama dengan fungsinya. Cross-Region kunci tidak didukung. Untuk membuat kunci terkelola pelanggan simetris, ikuti langkah-langkah untuk membuat kunci KMS enkripsi simetris di Panduan PengembangAWS Key Management Service .

Izin

Anda memberikan AWS KMS izin melalui kebijakan kunci kunci KMS.

Mengkonfigurasi kunci yang dikelola pelanggan memerlukan AWS KMS izin pada prinsipal yang membuat atau memperbarui fungsi. Memanggil fungsi tahan lama tidak memerlukan AWS KMS izin pada penelepon; Lambda melakukan enkripsi dengan prinsipal layanannya.

Kebijakan kunci

Kebijakan utama mengontrol akses ke kunci yang dikelola pelanggan Anda. Setiap kunci yang dikelola pelanggan harus memiliki persis satu kebijakan utama. Dalam kebijakan utama, hanya

`Resource: "*"` mengacu pada kunci KMS yang dilampirkan kebijakan, bukan ke semua kunci KMS di akun Anda.

Kebijakan kunci fungsi tahan lama memberikan setiap prinsipal hanya AWS KMS tindakan yang dibutuhkannya, dicakup oleh layanan dan fungsi ARN. Bagian berikut menjelaskan pernyataan, kondisi, dan contoh lengkap.

Pernyataan kebijakan yang diperlukan

Kebijakan ini memberikan kemampuan berikut:

- Aktifkan Izin pengguna IAM. Memberikan akses tanpa syarat root akun untuk mengelola kunci. Pernyataan ini tidak menggunakan kondisi karena pelingkupan itu akan mencegah Anda memutar, memperbarui, atau menghapus kunci.
- Izinkan Lambda menggunakan tombol ini untuk fungsi yang tahan lama. Memberikan kepala layanan `kms:GenerateDataKey` Lambda dan. `kms:Decrypt`
- Izinkan peran eksekusi fungsi untuk mendekripsi data eksekusi yang tahan lama. Memberikan peran eksekusi `kms:Decrypt` untuk membaca status dan memajukan eksekusi.
- Izinkan penulis fungsi untuk mendeskripsikan kunci ini. Memberikan `kms:DescribeKey` sehingga Lambda dapat memvalidasi bahwa kunci simetris dan diaktifkan selama atau. `CreateFunction` `UpdateFunctionConfiguration`
- Izinkan pembuat fungsi untuk memvalidasi kunci ini untuk fungsi tertentu. Hibah `kms:GenerateDataKey` dan `kms:Decrypt` Lambda dapat memvalidasi izin kunci dan akses dengan konteks enkripsi fungsi selama atau. `CreateFunction` `UpdateFunctionConfiguration` Ini mengonfirmasi kebijakan kunci menerima panggilan untuk fungsi khusus ini sebelum fungsi dibuat atau diperbarui.
- Izinkan operator eksekusi yang tahan lama. Memberikan operator `kms:Decrypt` untuk panggilan ke`GetDurableExecution`, `GetDurableExecutionHistory` dengan`IncludeExecutionData=true`,, `GetDurableExecutionStateStopDurableExecution`, dan API panggilan balik.

Sebagai praktik terbaik, gunakan prinsip terpisah untuk peran eksekusi, pembuat fungsi, dan operator eksekusi tahan lama sehingga setiap identitas hanya memiliki kemampuan yang dibutuhkannya.

Ketentuan kebijakan yang direkomendasikan

Kebijakan ini menggunakan kondisi berikut untuk cakupan akses bawah:

- kms: ViaService membatasi penggunaan kunci untuk permintaan yang dirutekan melalui Lambda. Menerapkan ini ke peran eksekusi, pembuat fungsi, dan pernyataan operator mencegah mereka menggunakan kunci secara langsung AWS KMS.
- aws: SourceArn dan aws: SourceAccount melindungi dari masalah wakil lintas layanan yang membingungkan. Lambda meneruskan nilai-nilai ini ketika memanggil AWS KMS menggunakan kredensyal layanannya sendiri. Kondisi ini hanya berlaku untuk pernyataan utama layanan Lambda. Peran eksekusi, pembuat fungsi, dan pernyataan operator memanggil AWS KMS dengan kredenal sesi akses penerusan pemanggil, yang tidak membawa header ini.
- kms: EncryptionContext:aws:lambda:FunctionArn mencakup kunci ke fungsi tertentu. Lambda menambahkan ini ke konteks enkripsi pada setiap AWS KMS panggilan untuk data eksekusi yang tahan lama.

Contoh berikut menggabungkan pernyataan dan kondisi di atas.

Example Kebijakan utama untuk fungsi yang tahan lama

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Enable IAM User Permissions",
            "Effect": "Allow",
            "Principal": { "AWS": "arn:aws:iam::111122223333:root" },
            "Action": "kms:*",
            "Resource": "*"
        },
        {
            "Sid": "Allow Lambda to use this key for durable functions",
            "Effect": "Allow",
            "Principal": { "Service": "lambda.amazonaws.com" },
            "Action": [ "kms:GenerateDataKey", "kms:Decrypt" ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "111122223333",
                    "aws:SourceArn": "arn:aws:lambda:us-
east-1:111122223333:function:myDurableFunction",
                    "kms:EncryptionContext:aws:lambda:FunctionArn": "arn:aws:lambda:us-
east-1:111122223333:function:myDurableFunction"
                }
```

```
            }
        },
        {
            "Sid": "Allow the function execution role to decrypt durable execution
 data",
            "Effect": "Allow",
            "Principal": { "AWS": "arn:aws:iam::111122223333:role/
myDurableFunctionRole" },
            "Action": "kms:Decrypt",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "kms:ViaService": "lambda.us-east-1.amazonaws.com",
                    "kms:EncryptionContext:aws:lambda:FunctionArn": "arn:aws:lambda:us-
east-1:111122223333:function:myDurableFunction"
                }
            }
        },
        {
            "Sid": "Allow the function author to describe this key",
            "Effect": "Allow",
            "Principal": { "AWS": "arn:aws:iam::111122223333:role/FunctionAuthor" },
            "Action": "kms:DescribeKey",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "kms:ViaService": "lambda.us-east-1.amazonaws.com"
                }
            }
        },
        {
            "Sid": "Allow the function author to validate this key for a specific
 function",
            "Effect": "Allow",
            "Principal": { "AWS": "arn:aws:iam::111122223333:role/FunctionAuthor" },
            "Action": [ "kms:GenerateDataKey", "kms:Decrypt" ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "kms:ViaService": "lambda.us-east-1.amazonaws.com",
                    "kms:EncryptionContext:aws:lambda:FunctionArn": "arn:aws:lambda:us-
east-1:111122223333:function:myDurableFunction"
                }
            }
```

```
        },
        {
            "Sid": "Allow durable execution operators",
            "Effect": "Allow",
            "Principal": { "AWS": "arn:aws:iam::111122223333:role/
DurableExecutionOperator" },
            "Action": "kms:Decrypt",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "kms:ViaService": "lambda.us-east-1.amazonaws.com",
                    "kms:EncryptionContext:aws:lambda:FunctionArn": "arn:aws:lambda:us-
east-1:111122223333:function:myDurableFunction"
                }
            }
        }
    ]
}
```

Untuk informasi selengkapnya tentang kebijakan AWS KMS utama, lihat Cara mengubah kebijakan kunci di Panduan AWS Key Management Service Pengembang.

Konfigurasikan kunci yang dikelola pelanggan pada fungsi yang tahan lama

Anda mengonfigurasi kunci terkelola pelanggan untuk data eksekusi yang tahan lama melalui `KMSKeyArn` bidang `DurableConfig` objek. Atur saat Anda membuat fungsi dengan CreateFunction; perbarui pada fungsi tahan lama yang ada dengan UpdateFunctionConfiguration.

Lambda console

Untuk mengonfigurasi kunci terkelola pelanggan pada fungsi tahan lama yang ada

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih fungsi yang tahan lama.
3. Pilih Konfigurasi, lalu pilih Eksekusi tahan lama dari bilah navigasi kiri.
4. Pilih Edit.
5. Di bawah Enkripsi, pilih Gunakan kunci yang dikelola pelanggan, lalu pilih kunci KMS dari daftar.
6. Pilih Simpan.

AWS CLI

Untuk mengonfigurasi kunci terkelola pelanggan saat Anda membuat fungsi

Dalam contoh fungsi pembuatan berikut, `--durable-config` opsi menentukan ARN kunci terkelola pelanggan bersama dengan batas waktu eksekusi dan periode retensi yang tahan lama.

```
aws lambda create-function \
  --function-name myDurableFunction \
  --runtime nodejs24.x \
  --handler index.handler \
  --role arn:aws:iam::111122223333:role/myDurableFunctionRole \
  --zip-file fileb://function.zip \
  --durable-config '{"KMSKeyArn":"arn:aws:kms:us-east-1:111122223333:key/key-
id","ExecutionTimeout":3600,"RetentionPeriodInDays":30}'
```

Untuk mengonfigurasi kunci terkelola pelanggan pada fungsi tahan lama yang ada

Gunakan perintah update-function-configuration. Sertakan semua `DurableConfig` bidang yang ingin Anda simpan, karena `--durable-config` menggantikan seluruh objek.

```
aws lambda update-function-configuration \
  --function-name myDurableFunction \
  --durable-config '{"KMSKeyArn":"arn:aws:kms:us-east-1:111122223333:key/key-
id","ExecutionTimeout":3600,"RetentionPeriodInDays":30}'
```

Untuk menghapus kunci yang dikelola pelanggan dari fungsi yang tahan lama

Hilangkan `KMSKeyArn` dari`--durable-config`. Eksekusi yang ada terus menggunakan kunci yang dikelola pelanggan yang mereka mulai. Eksekusi baru menggunakan enkripsi default sebagai gantinya.

```
aws lambda update-function-configuration \
  --function-name myDurableFunction \
  --durable-config '{"ExecutionTimeout":3600,"RetentionPeriodInDays":30}'
```

AWS CloudFormation

Dalam `AWS::Lambda::Function` sumber daya, atur `KMSKeyArn` properti dari`DurableConfig`:

```
Resources:
```

```
MyDurableFunction:
  Type: AWS::Lambda::Function
  Properties:
    FunctionName: myDurableFunction
    Runtime: nodejs24.x
    Handler: index.handler
    Role: !GetAtt MyDurableFunctionRole.Arn
    Code:
      ZipFile: |
        // Your durable function code
    DurableConfig:
      KMSKeyArn: "arn:aws:kms:us-east-1:111122223333:key/key-id"
      ExecutionTimeout: 3600
      RetentionPeriodInDays: 30
```

Important

Setiap eksekusi tahan lama menggunakan kunci yang dikelola pelanggan yang dikonfigurasi pada fungsi saat dimulai. Mengubah atau menghapus kunci hanya memengaruhi eksekusi yang dimulai setelah perubahan; eksekusi dalam penerbangan terus menggunakan kunci yang mereka mulai sampai selesai atau gagal.

Membaca data eksekusi yang tahan lama

Dua API baca mengembalikan data eksekusi yang tahan lama:

- `GetDurableExecution`mengembalikan keadaan saat ini dari eksekusi tunggal, termasuk muatan masukan, data pos pemeriksaan terbaru, dan informasi hasil atau kesalahan.
- `GetDurableExecutionHistory`mengembalikan daftar peristiwa yang diurutkan yang dipancarkan eksekusi, menunjukkan input dan hasil pos pemeriksaan, input dan hasil pemanggilan berantai, dan hasil akhir.

Kedua API menerima parameter `IncludeExecutionData` permintaan. `IncludeExecutionData`Kapan`true`, Lambda menggunakan kredensi pemanggil untuk memanggil kunci yang dikelola pelanggan`kms:Decrypt`. Lambda kemudian mengembalikan data eksekusi yang didekripsi dalam respons. Identitas penelepon harus ada `kms:Decrypt` pada kunci yang dikelola pelanggan.

Kapan `IncludeExecutionData` ada`false`, Lambda tidak memanggil AWS KMS atau mendekripsi data eksekusi, dan responsnya disetel ke. `ExecutionDataIncluded` `false` Tanggapan masih termasuk`DurableConfig`, yang menggemakan ARN kunci yang dikelola pelanggan yang digunakan eksekusi. `IncludeExecutionData`default ke`false`, jadi penelepon yang hanya membutuhkan metadata tidak memerlukan izin. AWS KMS

Contoh: **`GetDurableExecution`** respon dengan **`IncludeExecutionData=false`**

```
{
    "DurableExecutionArn": "arn:aws:lambda:us-
east-1:111122223333:function:myDurableFunction:execution:exec-abc123",
    "DurableExecutionName": "exec-abc123",
    "FunctionArn": "arn:aws:lambda:us-east-1:111122223333:function:myDurableFunction",
    "StartTimestamp": 1733256000,
    "Status": "RUNNING",
    "ExecutionDataIncluded": false,
    "DurableConfig": {
        "ExecutionTimeout": 3600,
        "KMSKeyArn": "arn:aws:kms:us-east-1:111122223333:key/key-id",
        "RetentionPeriodInDays": 30
    }
}
```

Panggilan dirantai

Ketika fungsi tahan lama menggunakan pemanggilan berantai untuk memanggil fungsi tahan lama lainnya, Lambda memperlakukan eksekusi turunan sebagai eksekusi tahan lama yang independen: ia memiliki ID eksekusi sendiri, aliran pos pemeriksaannya sendiri, dan kunci yang dikelola pelanggannya sendiri. Konfigurasi kunci terkelola pelanggan induk tidak berpengaruh pada data anak, dan konfigurasi kunci terkelola pelanggan anak tidak berpengaruh pada data induk.

Izin

Kepala layanan Lambda harus memiliki `kms:GenerateDataKey` dan `kms:Decrypt` pada kunci yang dikelola pelanggan fungsi anak. Jika fungsi induk dan anak menggunakan kunci terkelola pelanggan yang berbeda, Anda memberikan akses utama layanan pada setiap kunci secara terpisah. Peran eksekusi fungsi induk tidak memerlukan izin pada kunci terkelola pelanggan anak.

Mengidentifikasi kunci mana yang dienkripsi eksekusi mana

Keduanya `GetDurableExecution` dan `ListDurableExecutionsByFunction` mengembalikan ARN kunci terkelola pelanggan yang mengenkripsi setiap eksekusi. Gunakan API ini untuk mengonfirmasi kunci mana yang berlaku saat eksekusi induk atau anak dimulai.

Caching kunci data

Untuk mengurangi volume AWS KMS panggilan dan meningkatkan ketersediaan selama gangguan layanan, Lambda menyimpan kunci data yang dihasilkan dari kunci yang dikelola pelanggan Anda hingga 15 menit. Sementara cache hangat, Lambda menggunakan kembali kunci data yang sama di seluruh operasi dalam eksekusi dan di seluruh eksekusi dimulai selama jendela cache.

Biaya

Per-execution AWS KMS biaya tetap rendah karena Lambda memanggil paling banyak `GenerateDataKey` sekali per jendela cache, tidak sekali per pos pemeriksaan. Pada tingkat permintaan tinggi, biaya per eksekusi turun lebih jauh karena lebih banyak eksekusi berbagi setiap kunci data yang di-cache. Anda ditagih untuk AWS KMS panggilan yang sebenarnya dilakukan Lambda, bukan untuk setiap pos pemeriksaan atau pembacaan.

Stabilitas statis

Kunci data tetap di-cache hingga 15 menit. Perubahan pada kunci Anda akan berlaku pada penyegaran cache berikutnya. Selama gangguan layanan yang diperpanjang, Lambda terus melayani dari cache, menjaga eksekusi dalam penerbangan tetap berjalan.

CloudTrail

Caching kunci data berarti Anda melihat lebih sedikit `GenerateDataKey` peristiwa CloudTrail daripada eksekusi atau pos pemeriksaan. Lihat Memantau kunci KMS untuk fungsi yang tahan lama misalnya acara.

Ketika kunci yang dikelola pelanggan tidak tersedia

Jika kunci yang dikelola pelanggan yang memulai eksekusi dinonaktifkan, dijadwalkan untuk dihapus, atau aksesnya dicabut melalui kebijakan kunci, eksekusi tidak dapat membuat kemajuan lebih lanjut. Di pos pemeriksaan berikutnya, Lambda gagal eksekusi dengan kesalahan yang tidak dapat dicoba ulang. AWS KMS Memulihkan akses ke kunci tidak secara otomatis melanjutkan eksekusi. Anda harus memulai eksekusi baru.

API yang membaca atau menulis muatan juga gagal saat kunci tidak tersedia. Mereka dapat mengembalikan salah satu pengecualian berikut:

- `KMSAccessDeniedException`: Lambda tidak dapat mendekripsi data eksekusi yang tahan lama karena akses ke kunci KMS ditolak.
- `KMSDisabledException`: Lambda tidak dapat mendekripsi data eksekusi yang tahan lama karena kunci KMS dinonaktifkan.
- `KMSInvalidStateException`: Lambda tidak dapat mendekripsi data eksekusi yang tahan lama karena status kunci KMS tidak valid untuk. `Decrypt`
- `KMSNotFoundException`: Lambda tidak dapat mendekripsi data eksekusi yang tahan lama karena kunci KMS tidak ditemukan.

Memulihkan akses ke kunci KMS memungkinkan API baca ini berhasil lagi untuk eksekusi yang sudah gagal. Eksekusi gagal tetap gagal; Anda harus memulai eksekusi baru.

Untuk memeriksa metadata eksekusi saat kunci KMS tidak tersedia, panggil dengan. `GetDurableExecution` `IncludeExecutionData=false` Ini mengembalikan status eksekusi, stempel waktu, dan `DurableConfig` (termasuk ARN kunci KMS) tanpa memanggil. AWS KMS

Karena Lambda menyimpan kunci data, menonaktifkan kunci tidak segera berlaku. Lihat perinciannya di Caching kunci data.

> **Important**
>
> Menghapus kunci KMS yang eksekusi dalam penerbangan atau dipertahankan masih bergantung pada secara permanen menghancurkan eksekusi tersebut dan sejarahnya.

Memantau kunci KMS untuk fungsi yang tahan lama

Saat Anda menggunakan kunci yang dikelola pelanggan dengan fungsi tahan lama, Anda dapat menggunakannya AWS CloudTrailuntuk melacak AWS KMS panggilan yang dilakukan Lambda atas nama Anda. Untuk panduan umum tentang AWS KMS peristiwa penelusuran, lihat Menelusuri aktivitas AWS KMS API.

Untuk mengonfirmasi bahwa Lambda menggunakan kunci Anda sebagaimana dimaksud, cari bidang ini di setiap acara:

- `eventName`: salah satu`GenerateDataKey`,`Decrypt`, atau `DescribeKey`
- `eventSource`: `kms.amazonaws.com`

- `userIdentity.invokedBy`: `lambda.amazonaws.com`
- `requestParameters.encryptionContext.aws:lambda:FunctionArn`: ARN dari fungsi tahan lama

Contoh berikut adalah CloudTrail peristiwa dari `UpdateFunctionConfiguration` panggilan `CreateFunction` atau yang mengonfigurasi kunci yang dikelola pelanggan pada fungsi yang tahan lama. Lambda mengeluarkan tiga AWS KMS panggilan untuk memvalidasi kunci: real`DescribeKey`, dan dry-run dan. `GenerateDataKey` `Decrypt`

GenerateDataKey

Saat Anda menyetel atau mengubah`DurableConfig`, Lambda mengeluarkan dry-run `GenerateDataKey` terhadap kunci terkelola pelanggan Anda untuk memvalidasi bahwa kebijakan kunci memungkinkan Lambda memperoleh kunci data untuk konteks enkripsi fungsi ini. `KMSKeyArn` Dry-run tidak melakukan pekerjaan kriptografi tetapi merekam peristiwa lengkap. CloudTrail Contoh peristiwa berikut mencatat operasi dry-run`GenerateDataKey`:


```
{
    "eventVersion": "1.11",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROA123456789EXAMPLE:example",
        "arn": "arn:aws:sts::111122223333:assumed-role/FunctionAuthor/example",
        "accountId": "111122223333",
        "accessKeyId": "ASIA123456789EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROA123456789EXAMPLE",
                "arn": "arn:aws:iam::111122223333:role/FunctionAuthor",
                "accountId": "111122223333",
                "userName": "FunctionAuthor"
            },
            "attributes": {
                "creationDate": "2026-01-15T16:54:42Z",
                "mfaAuthenticated": "false"
            }
        },
        "invokedBy": "lambda.amazonaws.com"
    },
    "eventTime": "2026-01-15T16:55:00Z",
```

```
    "eventSource": "kms.amazonaws.com",
    "eventName": "GenerateDataKey",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "lambda.amazonaws.com",
    "userAgent": "lambda.amazonaws.com",
    "errorCode": "DryRunOperationException",
    "errorMessage": "The request would have succeeded, but the DryRun option is
 set.",
    "requestParameters": {
        "encryptionContext": {
            "aws:lambda:FunctionArn": "arn:aws:lambda:us-
east-1:111122223333:function:myDurableFunction"
        },
        "dryRun": true,
        "keyId": "arn:aws:kms:us-east-1:111122223333:key/key-id",
        "keySpec": "AES_256"
    },
    "responseElements": null,
    "additionalEventData": {
        "keyMaterialId":
 "a1b2c3d4e5f6a7b8c9d0e1f2a3b4c5d6e7f8a9b0c1d2e3f4a5b6c7d8e9f0EXAMPLE"
    },
    "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEaaaaa",
    "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbb",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-east-1:111122223333:key/key-id"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

Decrypt

Saat Anda menyetel atau mengubah `KMSKeyArn` in`DurableConfig`, Lambda juga mengeluarkan dry-run `Decrypt` terhadap kunci terkelola pelanggan Anda untuk memvalidasi bahwa kebijakan kunci memungkinkan Lambda mendekripsi data untuk konteks enkripsi fungsi

ini. Penggunaan dry-run`IGNORE_CIPHERTEXT`, jadi tidak diperlukan ciphertext nyata. Contoh peristiwa berikut mencatat operasi dry-run`Decrypt`:


```
{
    "eventVersion": "1.11",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROA123456789EXAMPLE:example",
        "arn": "arn:aws:sts::111122223333:assumed-role/FunctionAuthor/example",
        "accountId": "111122223333",
        "accessKeyId": "ASIA123456789EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROA123456789EXAMPLE",
                "arn": "arn:aws:iam::111122223333:role/FunctionAuthor",
                "accountId": "111122223333",
                "userName": "FunctionAuthor"
            },
            "attributes": {
                "creationDate": "2026-01-15T16:54:42Z",
                "mfaAuthenticated": "false"
            }
        },
        "invokedBy": "lambda.amazonaws.com"
    },
    "eventTime": "2026-01-15T16:55:00Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "Decrypt",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "lambda.amazonaws.com",
    "userAgent": "lambda.amazonaws.com",
    "errorCode": "DryRunOperationException",
    "errorMessage": "The request would have succeeded, but the DryRun option is
 set.",
    "requestParameters": {
        "encryptionContext": {
            "aws:lambda:FunctionArn": "arn:aws:lambda:us-
east-1:111122223333:function:myDurableFunction"
        },
        "dryRun": true,
        "keyId": "arn:aws:kms:us-east-1:111122223333:key/key-id",
        "dryRunModifiers": ["IGNORE_CIPHERTEXT"],
```

```
        "encryptionAlgorithm": "SYMMETRIC_DEFAULT"
    },
    "responseElements": null,
    "additionalEventData": {
        "keyMaterialId":
 "a1b2c3d4e5f6a7b8c9d0e1f2a3b4c5d6e7f8a9b0c1d2e3f4a5b6c7d8e9f0EXAMPLE"
    },
    "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEaaaaa",
    "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbb",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-east-1:111122223333:key/key-id"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```

DescribeKey

Saat Anda mengatur atau mengubah `KMSKeyArn` in`DurableConfig`, Lambda memanggil `DescribeKey` untuk mengonfirmasi bahwa kunci simetris dan diaktifkan sebelum menerima perubahan. Berbeda dengan `GenerateDataKey` dan `Decrypt` probe, ini adalah panggilan nyata, bukan dry-run. Contoh peristiwa berikut mencatat `DescribeKey` operasi:

```
{
    "eventVersion": "1.11",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROA123456789EXAMPLE:example",
        "arn": "arn:aws:sts::111122223333:assumed-role/FunctionAuthor/example",
        "accountId": "111122223333",
        "accessKeyId": "ASIA123456789EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROA123456789EXAMPLE",
                "arn": "arn:aws:iam::111122223333:role/FunctionAuthor",
```

```
                "accountId": "111122223333",
                "userName": "FunctionAuthor"
            },
            "attributes": {
                "creationDate": "2026-01-15T16:54:42Z",
                "mfaAuthenticated": "false"
            }
        },
        "invokedBy": "lambda.amazonaws.com"
    },
    "eventTime": "2026-01-15T16:55:00Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "DescribeKey",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "lambda.amazonaws.com",
    "userAgent": "lambda.amazonaws.com",
    "requestParameters": {
        "keyId": "arn:aws:kms:us-east-1:111122223333:key/key-id"
    },
    "responseElements": null,
    "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEaaaaa",
    "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbb",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-east-1:111122223333:key/key-id"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```


Topik enkripsi terkait

- Enkripsi data saat istirahat untuk Lambda
- Mengamankan variabel lingkungan Lambda
- Mengenkripsi paket deployment .zip Lambda

- Model keamanan untuk Lambda SnapStart
- Konfigurasikan fungsi Lambda yang tahan lama

SDK eksekusi tahan lama

SDK eksekusi tahan lama adalah fondasi untuk membangun fungsi yang tahan lama. Ini menyediakan primitif untuk pemeriksaan kemajuan, menangani percobaan ulang, dan mengelola aliran eksekusi. SDK mengabstraksi kompleksitas manajemen pos pemeriksaan dan pemutaran ulang, memungkinkan Anda menulis kode berurutan yang secara otomatis menjadi toleran terhadap kesalahan.

SDK tersedia untuk JavaScript, TypeScript, Python, Java, Go, dan C # (.NET). Untuk dokumentasi API lengkap, tutorial memulai cepat, dan panduan khusus bahasa, lihat AWS Panduan Pengembang SDK Eksekusi Tahan Lama.

Apa yang dilakukan SDK

Manajemen pos pemeriksaan: SDK secara otomatis membuat pos pemeriksaan saat fungsi Anda menjalankan operasi yang tahan lama. Setiap pos pemeriksaan mencatat jenis operasi, input, dan hasil. Ketika fungsi Anda menyelesaikan satu langkah, SDK akan mempertahankan pos pemeriksaan sebelum melanjutkan. Ini memastikan fungsi Anda dapat melanjutkan dari operasi yang selesai jika terganggu.

Koordinasi putar ulang: Saat fungsi Anda dilanjutkan setelah jeda atau interupsi, SDK akan melakukan pemutaran ulang. Ini menjalankan kode Anda dari awal tetapi melewatkan operasi yang telah selesai, menggunakan hasil pos pemeriksaan yang disimpan alih-alih mengeksekusinya kembali. SDK memastikan pemutaran ulang bersifat deterministik. Dengan input dan log pos pemeriksaan yang sama, fungsi Anda menghasilkan hasil yang sama.

Isolasi negara: SDK mempertahankan status eksekusi secara terpisah dari logika bisnis Anda. Setiap eksekusi tahan lama memiliki log pos pemeriksaan sendiri yang tidak dapat diakses oleh eksekusi lain. SDK mengenkripsi data pos pemeriksaan saat diam dan memastikan status tetap konsisten di seluruh pemutaran ulang.

Untuk penjelasan rinci tentang cara kerja checkpointing dan perilaku pemutaran ulang, lihat Konsep kunci di AWS Panduan Pengembang SDK Eksekusi Tahan Lama.

Operasi tahan gangguan

SDK menyediakan fungsi Anda dengan `DurableContext` objek. Konteks ini menggantikan konteks Lambda standar dan menyediakan metode untuk membuat pos pemeriksaan, mengelola alur eksekusi, dan berkoordinasi dengan sistem eksternal.

`DurableContext` menyediakan operasi berikut untuk membangun alur kerja yang tahan lama:

Operasi	Deskripsi
Langkah	Jalankan dan periksa unit kerja dengan strategi coba ulang yang dapat dikonfigurasi dan semantik eksekusi.
Tunggu	Jeda eksekusi untuk durasi tertentu tanpa menghabiskan sumber daya komputasi.
Tunggu Kondisi	Jajak pendapat untuk kondisi dengan checkpoint otomatis di antara upaya.
Panggilan balik	Jeda eksekusi dan tunggu sistem eksternal memberikan masukan melalui API Lambda.
Memanggil	Panggil fungsi Lambda lain dan tunggu hasilnya, dengan checkpoin ting otomatis.
Paralel	Jalankan beberapa operasi secara bersamaan dengan kebijakan penyelesaian yang dapat dikonfigurasi.
Peta	Memproses setiap item dalam koleksi secara bersamaan dengan kontrol konkurensi opsional.
Konteks Anak	Buat konteks eksekusi terisolasi untuk mengelompokkan beberapa operasi.

Setiap operasi yang tahan lama menciptakan pos pemeriksaan secara otomatis, memastikan fungsi Anda dapat dilanjutkan dari titik mana pun. Untuk referensi API terperinci, contoh kode, dan penggunaan khusus bahasa, lihat Referensi SDK di AWS Panduan Pengembang SDK Eksekusi Tahan Lama.

Bagaimana operasi yang tahan lama diukur

Setiap operasi tahan lama yang Anda hubungi `DurableContext` membuat pos pemeriksaan untuk melacak kemajuan eksekusi dan menyimpan data status. Operasi ini dikenakan biaya berdasarkan penggunaannya, dan pos pemeriksaan mungkin berisi data yang berkontribusi pada biaya penulisan dan penyimpanan data Anda. Data yang disimpan termasuk data peristiwa pemanggilan, muatan yang dikembalikan dari langkah-langkah, dan data yang diteruskan saat menyelesaikan panggilan balik. Memahami bagaimana operasi yang tahan lama diukur membantu Anda memperkirakan biaya eksekusi dan mengoptimalkan alur kerja Anda. Untuk detail tentang harga, lihat Halaman harga Lambda.

Ukuran muatan mengacu pada ukuran data serial yang dipertahankan oleh operasi yang tahan lama. Data diukur dalam byte dan ukurannya dapat bervariasi tergantung pada serializer yang digunakan oleh operasi. Muatan operasi bisa menjadi hasil itu sendiri untuk penyelesaian yang berhasil, atau objek kesalahan serial jika operasi gagal.

Operasi dasar

Operasi dasar adalah blok bangunan mendasar untuk fungsi tahan lama:

Operasi	Waktu pos pemeriksaan	Jumlah operasi	Data tetap ada
Eksekusi	Dimulai	1	Ukuran muatan masukan
Eksekusi	Selesai (Succeeded/Failed/Stopped)	0	Ukuran muatan keluaran
Langkah	Retry/Succeeded/Failed	1 + N percobaan ulang	Mengembalikan ukuran muatan dari setiap upaya
Tunggu	Dimulai	1	N/A
WaitForCondition	Setiap upaya jajak pendapat	1 + N jajak pendapat	Mengembalikan ukuran muatan dari setiap upaya jajak pendapat

Operasi	Waktu pos pemeriksaan	Jumlah operasi	Data tetap ada
Invocation-level Coba lagi	Dimulai	1	Payload untuk objek kesalahan

Operasi panggilan balik

Operasi panggilan balik memungkinkan fungsi Anda untuk menjeda dan menunggu sistem eksternal memberikan masukan. Operasi ini membuat pos pemeriksaan saat panggilan balik dibuat dan setelah selesai:

Operasi	Waktu pos pemeriksaan	Jumlah operasi	Data tetap ada
CreateCallback	Dimulai	1	N/A
Penyelesaian panggilan balik melalui panggilan API	Selesai	0	Muatan panggilan balik
WaitForCallback	Dimulai	3 + N percobaan ulang (konteks + panggilan balik + langkah)	Payload dikembalikan oleh upaya langkah pengirim, ditambah dua salinan payload callback

Operasi majemuk

Operasi gabungan menggabungkan beberapa operasi tahan lama untuk menangani pola koordinasi yang kompleks seperti eksekusi paralel, pemrosesan array, dan konteks bersarang:

Operasi	Waktu pos pemeriksaan	Jumlah operasi	Data tetap ada
Paralel	Dimulai	1 + N cabang (1 konteks induk + N konteks anak)	Hingga dua salinan ukuran muatan yang dikembalikan dari setiap cabang, ditambah status setiap cabang
Peta	Dimulai	1 + N cabang (1 konteks induk + N konteks anak)	Hingga dua salinan ukuran payload yang dikembalikan dari setiap iterasi, ditambah status setiap iterasi
Pembantu janji	Selesai	1	Ukuran muatan yang dikembalikan dari janji
RunInChildContext	Succeeded/Failed	1	Mengembalikan ukuran payload dari konteks anak

Untuk konteks, seperti dari `runInChildContext` atau digunakan secara internal oleh operasi gabungan, hasil yang lebih kecil dari 256 KB diperiksa secara langsung. Hasil yang lebih besar tidak disimpan — sebaliknya, mereka direkonstruksi selama pemutaran ulang dengan memproses ulang operasi konteks.

Runtime yang didukung untuk fungsi yang tahan lama

Fungsi tahan lama tersedia untuk runtime terkelola yang dipilih dan gambar wadah OCI untuk fleksibilitas versi runtime tambahan. Anda dapat membuat fungsi tahan lama untuk Node.js, Python, Java, dan C# (.NET) menggunakan runtime terkelola langsung di konsol atau secara terprogram melalui infrastruktur-as-code.

Runtime yang dikelola Lambda

Runtime terkelola berikut mendukung fungsi tahan lama saat Anda membuat fungsi di konsol Lambda atau menggunakan AWS CLI dengan parameter. `--durable-config '{"ExecutionTimeout": 3600, "RetentionPeriodInDays": 7}'` Untuk informasi lengkap tentang runtime Lambda, lihat runtime Lambda.

Bahasa	Waktu Aktif
Node.js	nodejs22.x
Node.js	nodejs24.x
Python	python3.13
Python	python3.14
Java	java17
Java	java21
Java	java25
C# (.NET)	dotnet8
C# (.NET)	dotnet10

> **Note**
>
> Runtime Lambda Node.js dan Python menyertakan SDK eksekusi tahan lama untuk pengujian dan pengembangan. Namun, sebaiknya sertakan SDK dalam paket penerapan Anda untuk produksi. Ini memastikan konsistensi versi dan menghindari pembaruan runtime potensial yang mungkin memengaruhi perilaku fungsi Anda. Karena Java dan C# (.NET) adalah bahasa yang dikompilasi, runtime Lambda Java dan.NET tidak menyertakan SDK eksekusi tahan lama, sehingga harus disertakan dalam paket penerapan Anda.

Node.js

Instal SDK di Node.js proyek Anda:

```
npm install @aws/durable-execution-sdk-js
```

SDK mendukung JavaScript dan TypeScript. Untuk TypeScript proyek, SDK menyertakan definisi tipe.

Python

Instal SDK di proyek Python Anda:

```
pip install aws-durable-execution-sdk-python
```

Python SDK menggunakan metode sinkron dan tidak memerlukan`async/await`.

Java

Tambahkan ketergantungan kepom.xml:

```
<dependency>
    <groupId>software.amazon.lambda.durable</groupId>
    <artifactId>aws-durable-execution-sdk-java</artifactId>
    <version>VERSION</version>
</dependency>
```

Instal SDK di proyek Java Anda:

```
mvn install
```

Java SDK menyediakan versi sinkron dan asinkron dari setiap metode.

C# (.NET)

Tambahkan SDK ke proyek.NET Anda:

```
dotnet add package Amazon.Lambda.DurableExecution
```

.NET SDK mengekspos operasi yang tahan lama melalui`IDurableContext`.
Handler Anda menerima amplop layanan eksekusi tahan lama dan mendelegasikannya`DurableFunction.WrapAsync`. Setiap badan operasi membutuhkan`CancellationToken`.

Image kontainer

Anda dapat menggunakan fungsi tahan lama dengan gambar kontainer untuk mendukung versi runtime tambahan atau konfigurasi runtime khusus. Gambar kontainer memungkinkan Anda menggunakan versi runtime yang tidak tersedia sebagai runtime terkelola atau menyesuaikan lingkungan runtime Anda.

Untuk membuat fungsi tahan lama menggunakan gambar wadah:

1. Buat Dockerfile berdasarkan gambar dasar Lambda
2. Instal SDK eksekusi tahan lama di wadah Anda
3. Buat dan dorong gambar kontainer ke Amazon Elastic Container Registry
4. Buat fungsi Lambda dari gambar kontainer dengan eksekusi tahan lama diaktifkan

Contoh kontainer

Buat Dockerfile:

Python

Buat Dockerfile untuk Python 3.11:

```
FROM public.ecr.aws/lambda/python:3.11

# Copy requirements file
COPY requirements.txt ${LAMBDA_TASK_ROOT}/
```

```
# Install dependencies including durable SDK
RUN pip install -r requirements.txt

# Copy function code
COPY lambda_function.py ${LAMBDA_TASK_ROOT}/

# Set the handler
CMD [ "lambda_function.handler" ]
```

Buat `requirements.txt` file:

```
aws-durable-execution-sdk-python
```

Java

Buat Dockerfile untuk Java 25:

```
FROM --platform=linux/amd64 public.ecr.aws/lambda/java:25

# Install Maven
RUN dnf install -y maven

WORKDIR /var/task

# Copy Maven configuration and source code
COPY pom.xml .
COPY src ./src

# Build
RUN mvn clean package -DskipTests

# Move JAR to lib directory
RUN mv target/*.jar lib/

# Set the handler
CMD ["src.path.to.lambdaFunction::handler"]
```

Bangun dan dorong gambar:

```
# Build the image
docker build -t my-durable-function .

# Tag for ECR
docker tag my-durable-function:latest 123456789012.dkr.ecr.us-east-1.amazonaws.com/my-
durable-function:latest

# Push to ECR
docker push 123456789012.dkr.ecr.us-east-1.amazonaws.com/my-durable-function:latest
```

Buat fungsi dengan eksekusi tahan lama diaktifkan:

```
aws lambda create-function \
  --function-name myDurableFunction \
  --package-type Image \
  --code ImageUri=123456789012.dkr.ecr.us-east-1.amazonaws.com/my-durable-
function:latest \
  --role arn:aws:iam::123456789012:role/lambda-execution-role \
  --durable-config '{"ExecutionTimeout": 3600, "RetentionPeriodInDays": 7}'
```

Untuk informasi selengkapnya tentang menggunakan gambar kontainer dengan Lambda, lihat Membuat gambar wadah Lambda di Panduan Pengembang Lambda.

Pertimbangan runtime

Manajemen versi SDK: Untuk mempertahankan kontrol penuh atas dependensi Anda dan untuk menghindari kemungkinan masalah ketidakselarasan versi, kami sarankan Anda menambahkan semua dependensi fungsi Anda ke paket penerapan Anda, bahkan jika versinya disertakan dalam runtime Lambda secara default. Ini termasuk SDK eksekusi tahan lama. Kunci SDK eksekusi tahan lama ke versi utama di file ketergantungan Anda. Versi utama baru dapat memperkenalkan perubahan yang dapat mengakibatkan kegagalan eksekusi dalam penerbangan. Gunakan versi bernomor atau alias untuk fungsi produksi tahan lama daripada `$LATEST` memastikan perubahan versi SDK tidak memengaruhi eksekusi dalam penerbangan.

Pembaruan runtime: AWS memperbarui runtime terkelola untuk menyertakan patch keamanan dan perbaikan bug. Pembaruan ini mungkin menyertakan versi SDK baru. Untuk menghindari perilaku yang tidak terduga, sertakan SDK dalam paket penerapan Anda dan uji secara menyeluruh sebelum menerapkan ke produksi.

Ukuran gambar kon tainer: Gambar kontainer memiliki ukuran tidak terkompresi maksimum 10 GB. SDK eksekusi tahan lama menambahkan ukuran minimal ke gambar Anda. Optimalkan wadah Anda dengan menggunakan build multi-tahap dan menghapus dependensi yang tidak perlu.

Kinerja start dingin: Gambar kontainer mungkin memiliki waktu start dingin yang lebih lama daripada runtime yang dikelola. SDK eksekusi tahan lama memiliki dampak minimal pada kinerja cold start. Gunakan konkurensi yang disediakan jika latensi start dingin sangat penting untuk aplikasi Anda.

Memohon fungsi Lambda yang tahan lama

Fungsi Lambda yang tahan lama mendukung metode pemanggilan yang sama dengan fungsi Lambda lainnya. Anda dapat menjalankan fungsi tahan lama secara sinkron, asinkron, atau melalui pemetaan sumber peristiwa. Proses pemanggilan identik dengan fungsi lain, tetapi fungsi yang tahan lama memberikan kemampuan tambahan untuk eksekusi jangka panjang dan manajemen status otomatis.

Metode pemanggilan

Pemanggilan sinkron: Memanggil fungsi yang tahan lama dan tunggu respons. Pemanggilan sinkron dibatasi hingga 15 menit, atau kurang jika batas waktu fungsi atau batas waktu eksekusi lebih rendah. Gunakan pemanggilan sinkron saat Anda membutuhkan hasil segera atau saat mengintegrasikan dengan API dan layanan yang mengharapkan respons. Anda dapat menggunakan operasi tunggu untuk perhitungan yang efisien tanpa mengganggu pemanggil—pemanggilan menunggu seluruh eksekusi yang tahan lama selesai. Untuk eksekusi idempoten dimulai, gunakan parameter nama eksekusi seperti yang dijelaskan dalam Idempotency.

```
aws lambda invoke \
  --function-name my-durable-function:1 \
  --cli-binary-format raw-in-base64-out \
  --payload '{"orderId": "12345"}' \
  response.json
```

Pemanggilan asinkron: Antrikan acara untuk diproses tanpa menunggu respons. Lambda menempatkan acara dalam antrian dan segera kembali. Pemanggilan asinkron mendukung durasi eksekusi hingga 1 tahun. Gunakan pemanggilan asinkron untuk skenario fire-and-forget atau saat pemrosesan dapat terjadi di latar belakang. Untuk eksekusi idempoten dimulai, gunakan parameter nama eksekusi seperti yang dijelaskan dalam Idempotency.

```
aws lambda invoke \
  --function-name my-durable-function:1 \
  --invocation-type Event \
  --cli-binary-format raw-in-base64-out \
  --payload '{"orderId": "12345"}' \
  response.json
```

Pemetaan sumber peristiwa: Konfigurasikan Lambda untuk secara otomatis menjalankan fungsi tahan lama Anda saat rekaman tersedia dari layanan berbasis streaming atau antrian seperti Amazon SQS, Kinesis, atau DynamoDB. Pemetaan sumber peristiwa polling sumber acara dan menjalankan fungsi Anda dengan kumpulan catatan. Untuk detail tentang penggunaan pemetaan sumber peristiwa dengan fungsi tahan lama, termasuk batas durasi eksekusi, lihat Pemetaan sumber peristiwa dengan fungsi tahan lama.

Untuk detail lengkap tentang setiap metode pemanggilan, lihat pemanggilan sinkron dan pemanggilan asinkron.

> **Note**
>
> Fungsi tahan lama mendukung antrian huruf mati (DLQ) untuk penanganan kesalahan, tetapi tidak mendukung tujuan Lambda. Konfigurasikan DLQ untuk menangkap catatan dari pemanggilan yang gagal.

Persyaratan ARN yang memenuhi syarat

Fungsi yang tahan lama membutuhkan pengidentifikasi yang memenuhi syarat untuk pemanggilan. Anda harus menjalankan fungsi tahan lama menggunakan nomor versi, alias, atau. `$LATEST` Anda dapat menggunakan ARN yang memenuhi syarat penuh atau nama fungsi dengan version/alias akhiran. Anda tidak dapat menggunakan pengenal yang tidak memenuhi syarat (tanpa versi atau akhiran alias).

Doa yang valid:

```
# Using full ARN with version number
arn:aws:lambda:us-east-1:123456789012:function:my-durable-function:1

# Using full ARN with alias
arn:aws:lambda:us-east-1:123456789012:function:my-durable-function:prod

# Using full ARN with $LATEST
arn:aws:lambda:us-east-1:123456789012:function:my-durable-function:$LATEST

# Using function name with version number
my-durable-function:1

# Using function name with alias
my-durable-function:prod
```

Pemanggilan tidak valid:

```
# Unqualified ARN (not allowed)
arn:aws:lambda:us-east-1:123456789012:function:my-durable-function

# Unqualified function name (not allowed)
my-durable-function
```

Persyaratan ini memastikan bahwa eksekusi yang tahan lama tetap konsisten sepanjang siklus hidupnya. Ketika eksekusi tahan lama dimulai, itu disematkan ke versi fungsi yang memulainya. Jika fungsi Anda berhenti dan dilanjutkan beberapa jam atau hari kemudian, Lambda akan memanggil versi yang sama, jadi penerbitan versi baru hanya memengaruhi eksekusi baru. Demikian pula, ketika Anda memperbarui alias, eksekusi baru menggunakan versi baru, dan eksekusi yang sedang berlangsung berlanjut pada versi yang mereka mulai.

⚠ Eksekusi **$ TERBARU** dapat dilanjutkan pada kode yang diperbarui

Eksekusi yang dimulai dengan $LATEST qualifier tidak disematkan ke salinan tetap kode Anda. Jika Anda memperbarui kode fungsi saat eksekusi dijeda, eksekusi dilanjutkan pada kode baru daripada versi yang dimulai.

Kode Anda mungkin tidak lagi memproses status eksekusi yang disimpan dengan cara yang sama, yang menyebabkan kesalahan non-determinisme atau kegagalan diam. Untuk informasi selengkapnya tentang fungsi penulisan yang diputar ulang dengan aman, lihatVersi fungsi dan alias.

Praktik terbaik

Gunakan versi bernomor atau alias untuk fungsi tahan lama produksi daripada. `$LATEST` Versi bernomor tidak dapat diubah dan mendukung pemutaran ulang deterministik. Secara opsional, alias menyediakan referensi stabil yang dapat Anda perbarui untuk menunjuk ke versi baru tanpa mengubah kode pemanggilan. Gunakan `$LATEST` hanya untuk pembuatan prototipe atau untuk mempersingkat waktu penerapan selama pengembangan, karena eksekusi mungkin tidak diputar ulang dengan benar jika kode berubah saat dijalankan.

Memahami siklus hidup eksekusi

Saat Anda menjalankan fungsi tahan lama, Lambda membuat eksekusi tahan lama yang dapat menjangkau beberapa pemanggilan fungsi:

1. Pemanggilan awal: Permintaan pemanggilan Anda membuat eksekusi baru yang tahan lama. Lambda memberikan ID eksekusi unik dan mulai memproses.
2. Eksekusi dan pos pemeriksaan: Saat fungsi Anda menjalankan operasi yang tahan lama, SDK membuat pos pemeriksaan yang melacak kemajuan.
3. Penangguhan (jika diperlukan): Jika fungsi Anda menggunakan penantian yang tahan lama, seperti `wait` atau`waitForCallback`, atau percobaan ulang langkah otomatis, Lambda menangguhkan eksekusi dan menghentikan pengisian daya untuk waktu komputasi.
4. Dimulainya kembali: Ketika tiba waktunya untuk melanjutkan (termasuk setelah mencoba lagi), Lambda memanggil fungsi Anda lagi. SDK memutar ulang log pos pemeriksaan dan melanjutkan dari tempat eksekusi dijeda.
5. Penyelesaian: Ketika fungsi Anda mengembalikan hasil akhir atau melempar kesalahan yang tidak tertangani, eksekusi tahan lama selesai.

Untuk pemanggilan sinkron, penelepon menunggu seluruh eksekusi tahan lama selesai, termasuk operasi tunggu apa pun. Jika eksekusi melebihi batas waktu pemanggilan (15 menit atau kurang), waktu pemanggilan habis. Untuk pemanggilan asinkron, Lambda segera kembali dan eksekusi

berlanjut secara independen. Gunakan API eksekusi tahan lama untuk melacak status eksekusi dan mengambil hasil akhir.

Memohon dari kode aplikasi

Gunakan AWS SDK untuk menjalankan fungsi tahan lama dari kode aplikasi Anda, dengan `Invoke` API yang sama yang Anda gunakan untuk fungsi Lambda apa pun. Sertakan nomor versi, alias, atau `$LATEST` nama fungsi:

TypeScript

```
import { LambdaClient, InvokeCommand } from '@aws-sdk/client-lambda';

const client = new LambdaClient({});

// Synchronous invocation
const response = await client.send(new InvokeCommand({
  FunctionName: 'arn:aws:lambda:us-east-1:123456789012:function:my-durable-
function:1',
  Payload: JSON.stringify({ orderId: '12345' })
}));

const result = JSON.parse(Buffer.from(response.Payload!).toString());

// Asynchronous invocation
await client.send(new InvokeCommand({
  FunctionName: 'arn:aws:lambda:us-east-1:123456789012:function:my-durable-
function:1',
  InvocationType: 'Event',
  Payload: JSON.stringify({ orderId: '12345' })
}));
```

Python

```
import boto3
import json

client = boto3.client('lambda')
```

```
# Synchronous invocation
response = client.invoke(
    FunctionName='arn:aws:lambda:us-east-1:123456789012:function:my-durable-
function:1',
    Payload=json.dumps({'orderId': '12345'})
)

result = json.loads(response['Payload'].read())

# Asynchronous invocation
client.invoke(
    FunctionName='arn:aws:lambda:us-east-1:123456789012:function:my-durable-
function:1',
    InvocationType='Event',
    Payload=json.dumps({'orderId': '12345'})
)
```

Doa berantai

Fungsi tahan lama dapat memanggil fungsi Lambda apa pun, termasuk fungsi tahan lama lainnya, menggunakan operasi `invoke` dari. `DurableContext` Ini menciptakan pemanggilan berantai di mana fungsi pemanggilan ditangguhkan hingga fungsi yang dipanggil selesai:

TypeScript

```
export const handler = withDurableExecution(
  async (event: any, context: DurableContext) => {
    // Invoke another durable function and wait for result
    const result = await context.invoke(
      'process-order',
      'arn:aws:lambda:us-east-1:123456789012:function:order-processor:1',
      { orderId: event.orderId }
    );

    return { statusCode: 200, body: JSON.stringify(result) };
  }
);
```

Python

```
@durable_execution
def handler(event, context: DurableContext):
    # Invoke another durable function and wait for result
    result = context.invoke(
        'arn:aws:lambda:us-east-1:123456789012:function:order-processor:1',
        {'orderId': event['orderId']},
        name='process-order'
    )

    return {'statusCode': 200, 'body': json.dumps(result)}
```

Pemanggilan berantai membuat pos pemeriksaan dalam fungsi pemanggilan. Jika fungsi pemanggilan terputus, ia dilanjutkan dari pos pemeriksaan dengan hasil fungsi yang dipanggil, tanpa memanggil kembali fungsi tersebut.

> **Note**
>
> Cross-account pemanggilan berantai tidak didukung. Fungsi yang dipanggil harus berada di AWS akun yang sama dengan fungsi pemanggilan.

Pemetaan sumber acara dengan fungsi tahan lama

Fungsi tahan lama bekerja dengan semua pemetaan sumber acara Lambda. Konfigurasikan pemetaan sumber peristiwa untuk fungsi tahan lama dengan cara yang sama seperti Anda mengonfigurasinya untuk fungsi lain. Pemetaan sumber peristiwa secara otomatis melakukan polling sumber peristiwa seperti Amazon SQS, Kinesis, dan DynamoDB Streams, dan menjalankan fungsi Anda dengan kumpulan catatan.

Pemetaan sumber peristiwa berguna untuk fungsi tahan lama yang memproses aliran atau antrian dengan alur kerja multi-langkah yang kompleks. Misalnya, Anda dapat membuat fungsi tahan lama yang memproses pesan Amazon SQS dengan percobaan ulang, panggilan API eksternal, dan persetujuan manusia.

Bagaimana pemetaan sumber peristiwa memanggil fungsi yang tahan lama

Pemetaan sumber peristiwa memanggil fungsi tahan lama secara serempak, menunggu eksekusi tahan lama selesai sebelum memproses batch berikutnya atau menandai catatan sebagai diproses. Jika total waktu eksekusi tahan lama melebihi batas waktu tunggu yang berlaku (15 menit secara default, atau hingga 90 menit untuk fungsi yang berjalan di Instans Terkelola Lambda), eksekusi akan habis dan gagal. Pemetaan sumber peristiwa menerima pengecualian batas waktu dan menanganinya sesuai dengan konfigurasi coba ulang.

Batas durasi eksekusi

Jika fungsi tahan lama dipanggil oleh pemetaan sumber peristiwa, total durasi eksekusi tahan lama tidak dapat melebihi batas waktu fungsi maksimum: 15 menit untuk fungsi yang berjalan pada mode kapasitas default (sesuai permintaan), atau hingga 90 menit untuk fungsi yang berjalan pada Instans Terkelola Lambda (kecuali pemetaan sumber peristiwa Amazon MQ dan Amazon DocumentDB, yang tetap dibatasi hingga 15 menit). Batas ini berlaku untuk seluruh eksekusi tahan lama dari awal hingga penyelesaian, bukan hanya pemanggilan fungsi individual.

Batas waktu fungsi membatasi setiap pemanggilan. Batas waktu eksekusi yang tahan lama membatasi total waktu dari awal eksekusi hingga penyelesaian. Dengan pemetaan sumber peristiwa, keduanya memiliki maksimum yang sama: 15 menit secara default, atau 90 menit di Instans Terkelola Lambda.

Contoh skenario:

- Valid (mode kapasitas apa pun): Fungsi yang tahan lama memproses pesan Amazon SQS dengan tiga langkah, masing-masing memakan waktu 2 menit, lalu menunggu 5 menit sebelum menyelesaikan langkah terakhir. Total waktu eksekusi: 11 menit. Ini bekerja pada kedua mode kapasitas karena berada di bawah batas default 15 menit.
- Hanya berlaku untuk Instans Terkelola Lambda: Fungsi tahan lama memproses pesan Amazon SQS, menyelesaikan pemrosesan awal dalam 2 menit, lalu menunggu 20 menit untuk panggilan balik eksternal sebelum selesai. Total waktu eksekusi: 22 menit. Ini melebihi batas default 15 menit, tetapi berada dalam batas 90 menit untuk fungsi yang berjalan pada Instans Terkelola Lambda.
- Tidak valid (mode kapasitas apa pun): Fungsi yang tahan lama memproses catatan Kinesis dengan beberapa operasi tunggu dengan total dua jam di antara langkah-langkah. Ini bahkan melebihi batas Instans Terkelola Lambda 90 menit, sehingga eksekusi gagal terlepas dari mode kapasitas. Gunakan pola fungsi perantara untuk alur kerja selama ini.

> **Important**
>
> Konfigurasikan batas waktu eksekusi tahan lama Anda dalam batas yang berlaku saat menggunakan pemetaan sumber peristiwa, atau pembuatan pemetaan sumber peristiwa gagal. Batasnya adalah 15 menit secara default, atau hingga 90 menit pada Instans Terkelola Lambda. Jika alur kerja Anda membutuhkan waktu eksekusi yang lebih lama, gunakan pola fungsi perantara yang dijelaskan di bawah ini.

Mengkonfigurasi pemetaan sumber acara

Konfigurasikan pemetaan sumber peristiwa untuk fungsi tahan lama menggunakan konsol Lambda,, AWS CLI atau SDK. AWS Semua properti pemetaan sumber peristiwa, seperti ukuran batch dan jendela batching, berlaku untuk fungsi yang tahan lama:

```
aws lambda create-event-source-mapping \
  --function-name arn:aws:lambda:us-east-1:123456789012:function:my-durable-function:1
 \
  --event-source-arn arn:aws:sqs:us-east-1:123456789012:my-queue \
  --batch-size 10 \
  --maximum-batching-window-in-seconds 5
```

Gunakan ARN yang memenuhi syarat dengan nomor versi, alias, atau `$LATEST` saat Anda mengonfigurasi pemetaan sumber peristiwa untuk fungsi yang tahan lama.

Penanganan kesalahan dengan pemetaan sumber acara

Pemetaan sumber peristiwa menyediakan penanganan kesalahan bawaan yang berfungsi dengan fungsi tahan lama:

- Perilaku coba lagi: Jika pemanggilan awal gagal, pemetaan sumber peristiwa mencoba lagi sesuai dengan konfigurasi coba ulangnya. Konfigurasikan percobaan ulang maksimum dan interval coba lagi berdasarkan kebutuhan Anda.
- Dead-letter antrian: Konfigurasikan antrian huruf mati untuk menangkap catatan yang gagal setelah semua percobaan ulang. Ini mencegah kehilangan pesan dan memungkinkan pemeriksaan manual catatan yang gagal.

- Kegagalan batch sebagian: Untuk Amazon SQS dan Kinesis, gunakan pelaporan kegagalan batch sebagian untuk memproses rekaman satu per satu dan hanya mencoba lagi catatan yang gagal.
- Membagi dua kesalahan: Untuk Kinesis dan DynamoDB Streams, aktifkan bisect pada kesalahan untuk membagi batch yang gagal dan mengisolasi catatan bermasalah.

> **Note**
>
> Fungsi tahan lama mendukung antrian huruf mati (DLQ) untuk penanganan kesalahan, tetapi tidak mendukung tujuan Lambda. Konfigurasikan DLQ untuk menangkap catatan dari pemanggilan yang gagal.

Untuk informasi selengkapnya tentang penanganan kesalahan pemetaan sumber peristiwa, lihat pemetaan sumber peristiwa.

Menggunakan fungsi perantara untuk alur kerja yang berjalan lama

Jika alur kerja Anda memerlukan lebih dari batas pemetaan sumber peristiwa yang berlaku (15 menit secara default, atau 90 menit pada Instans Terkelola Lambda), gunakan fungsi Lambda standar perantara antara pemetaan sumber peristiwa dan fungsi tahan lama Anda. Fungsi perantara menerima peristiwa dari pemetaan sumber peristiwa dan memanggil fungsi tahan lama secara asinkron, menghapus batas durasi eksekusi.

Pola ini memisahkan model pemanggilan sinkron pemetaan sumber peristiwa dari model eksekusi jangka panjang fungsi tahan lama. Pemetaan sumber peristiwa memanggil fungsi perantara, yang dengan cepat kembali setelah memulai eksekusi yang tahan lama. Fungsi tahan lama kemudian berjalan secara mandiri selama dibutuhkan (hingga 1 tahun).

Arsitektur

Pola fungsi perantara menggunakan tiga komponen:

1. Pemetaan sumber peristiwa: Polling sumber peristiwa (Amazon SQS, Kinesis, DynamoDB Streams) dan memanggil fungsi perantara secara serempak dengan kumpulan catatan.
2. Fungsi perantara: Fungsi Lambda tanpa eksekusi tahan lama yang menerima peristiwa dari pemetaan sumber peristiwa, memvalidasi dan mengubah data jika diperlukan, dan memanggil fungsi tahan lama secara asinkron. Fungsi ini selesai dengan cepat (biasanya di bawah 1 detik) dan mengembalikan kontrol ke pemetaan sumber peristiwa.

3. Fungsi tahan lama: Memproses acara dengan logika multi-langkah yang kompleks yang dapat berjalan untuk waktu yang lama. Dipanggil secara asinkron, sehingga tidak dibatasi oleh batas durasi eksekusi pemetaan sumber peristiwa.

Implementasi

Fungsi perantara menerima seluruh peristiwa dari pemetaan sumber peristiwa dan memanggil fungsi tahan lama secara asinkron. Gunakan parameter nama eksekusi untuk memastikan eksekusi idempoten dimulai, mencegah pemrosesan duplikat jika pemetaan sumber peristiwa mencoba lagi:

TypeScript

```
import { LambdaClient, InvokeCommand } from '@aws-sdk/client-lambda';
import { SQSEvent } from 'aws-lambda';
import { createHash } from 'crypto';

const lambda = new LambdaClient({});

export const handler = async (event: SQSEvent) => {
  // Invoke durable function asynchronously with execution name
  await lambda.send(new InvokeCommand({
    FunctionName: 'arn:aws:lambda:us-east-1:123456789012:function:my-durable-
function:1',
    InvocationType: 'Event',
    Payload: JSON.stringify({
      executionName: event.Name,
      event: event
    })
  }));

  return { statusCode: 200 };
};
```

Python

```
import boto3
import json
import hashlib
```

```
lambda_client = boto3.client('lambda')

def handler(event, context):
    # Invoke durable function asynchronously with execution name
    lambda_client.invoke(
        FunctionName='arn:aws:lambda:us-east-1:123456789012:function:my-durable-
function:1',
        InvocationType='Event',
        Payload=json.dumps({
            'executionName': execution_name,
            'event': event["name"]
        })
    )

    return {'statusCode': 200}
```

Untuk idempotensi dalam fungsi perantara itu sendiri, gunakan Powertools AWS Lambda untuk mencegah pemanggilan duplikat dari fungsi tahan lama jika pemetaan sumber peristiwa mencoba kembali fungsi perantara.

Fungsi tahan lama menerima muatan dengan nama eksekusi dan memproses semua catatan dengan logika yang berjalan lama:

TypeScript

```
import { withDurableExecution, DurableContext } from '@aws/durable-execution-sdk-
js';

export const handler = withDurableExecution(
  async (payload: any, context: DurableContext) => {
    const sqsEvent = payload.event;

    // Process each record with complex, multi-step logic
    const results = await context.map(
      sqsEvent.Records,
      async (ctx, record) => {
        const validated = await ctx.step('validate', async () => {
          return validateOrder(JSON.parse(record.body));
        });
```

```
        // Wait for external approval (could take hours or days)
        const approval = await ctx.waitForCallback(
          'approval',
          async (callbackId) => {
            await requestApproval(callbackId, validated);
          },
          { timeout: { hours: 48 } }
        );

        // Complete processing
        return await ctx.step('complete', async () => {
          return completeOrder(validated, approval);
        });
      }
    );

    return { statusCode: 200, processed: results.getResults().length };
  }
);
```

Python

```
from aws_durable_execution_sdk_python import durable_execution, DurableContext
from aws_durable_execution_sdk_python.config import Duration, WaitForCallbackConfig
from collections.abc import Sequence
import json

def validate_order(order_data: dict) -> dict:
    """Validate order data - always passes."""
    return order_data

def request_approval(callback_id: str, validated_order: dict) -> None:
    """Request approval for the order - always passes."""
    pass

def complete_order(validated_order: dict, approval_result: str) -> dict:
    """Complete the order processing - always passes."""
    return validated_order

@durable_execution
def lambda_handler(payload, context: DurableContext):
```

```
    sqs_event = payload['event']

    def process_record(
        ctx: DurableContext,
        record: dict,
        index: int,
        items: Sequence[dict]
    ) -> dict:
        validated = ctx.step(
            lambda _: validate_order(json.loads(record['body'])),
            name=f'validate-{index}'
        )

        approval = ctx.wait_for_callback(
            submitter=lambda callback_id, wait_ctx: request_approval(callback_id,
validated),
            name=f'approval-{index}',
            config=WaitForCallbackConfig(timeout=Duration.from_seconds(172800))
        )

        return ctx.step(
            lambda _: complete_order(validated, approval),
            name=f'complete-{index}'
        )

    results = context.map(
        inputs=sqs_event['Records'],
        func=process_record,
        name='process-records'
    )

    return {
        'statusCode': 200,
        'started': results.started_count,
        'completed': results.success_count,
        'failed': results.failure_count,
        'total': results.total_count
    }
```

Pertimbangan utama

Pola ini menghapus batas durasi eksekusi pemetaan sumber peristiwa dengan memisahkan pemetaan sumber peristiwa dari eksekusi tahan lama. Fungsi perantara kembali segera setelah memulai eksekusi tahan lama, memungkinkan pemetaan sumber peristiwa untuk melanjutkan pemrosesan. Fungsi tahan lama kemudian berjalan secara independen selama dibutuhkan.

Fungsi perantara berhasil ketika memanggil fungsi tahan lama, bukan ketika eksekusi yang tahan lama selesai. Jika eksekusi tahan lama gagal nanti, pemetaan sumber peristiwa tidak akan mencoba lagi karena sudah berhasil memproses batch. Terapkan penanganan kesalahan dalam fungsi tahan lama dan konfigurasikan antrian huruf mati untuk eksekusi yang gagal.

Gunakan parameter nama eksekusi untuk memastikan eksekusi idempoten dimulai. Jika pemetaan sumber peristiwa mencoba ulang fungsi perantara, fungsi tahan lama tidak akan memulai eksekusi duplikat karena nama eksekusi sudah ada.

Sumber acara yang didukung

Fungsi tahan lama mendukung semua sumber acara Lambda yang menggunakan pemetaan sumber peristiwa:

- Antrian Amazon SQS (standar dan FIFO)
- Aliran Kinesis
- DynamoDB Streams
- Amazon Managed Streaming for Apache Kafka (Amazon MSK)
- Self-managed Apache Kafka
- Amazon MQ (ActiveMQ dan RabbitMQ)
- Amazon DocumentDB mengubah aliran

Semua jenis sumber peristiwa tunduk pada batas eksekusi tahan lama yang dijelaskan sebelumnya (15 menit secara default) saat menjalankan fungsi tahan lama. Pada Instans Terkelola Lambda, batas meningkat menjadi 90 menit untuk sumber acara ini, kecuali Amazon MQ dan Amazon DocumentDB, yang tetap dibatasi hingga 15 menit.

Mencoba lagi untuk fungsi tahan lama Lambda

Fungsi tahan lama memberikan kemampuan coba ulang otomatis yang membuat aplikasi Anda tahan terhadap kegagalan sementara. SDK menangani percobaan ulang pada dua tingkat: percobaan

ulang langkah untuk kegagalan logika bisnis dan percobaan ulang backend untuk kegagalan infrastruktur.

Langkah mencoba lagi

Ketika pengecualian yang tidak tertangkap terjadi dalam satu langkah, SDK secara otomatis mencoba ulang langkah berdasarkan strategi coba lagi yang dikonfigurasi. Langkah percobaan ulang adalah operasi checkpoint yang memungkinkan SDK untuk menangguhkan eksekusi dan melanjutkan nanti tanpa kehilangan kemajuan.

Langkah coba lagi perilaku

Tabel berikut menjelaskan cara SDK menangani pengecualian dalam langkah-langkah:

Skenario	Apa yang terjadi	Dampak pengukuran
Pengecualian sejalan dengan upaya coba lagi yang tersisa	SDK membuat pos pemeriksaan untuk coba lagi dan menangguhkan fungsi. Pada pemanggilan berikutnya, langkah mencoba lagi dengan penundaan backoff yang dikonfigurasi.	1 operasi+ukuran muatan kesalahan
Pengecualian dalam langkah tanpa upaya coba lagi yang tersisa	Langkah gagal dan melempar pengecualian. Jika kode handler Anda tidak menangkap pengecualian ini, seluruh eksekusi gagal.	1 operasi+ukuran muatan kesalahan

Ketika sebuah langkah perlu dicoba lagi, SDK memeriksa status coba lagi dan keluar dari pemanggilan Lambda jika tidak ada pekerjaan lain yang berjalan. Hal ini memungkinkan SDK untuk menerapkan penundaan backoff tanpa menghabiskan sumber daya komputasi. Fungsi dilanjutkan secara otomatis setelah periode backoff.

Mengkonfigurasi strategi coba lagi langkah

Konfigurasikan strategi coba lagi untuk mengontrol cara langkah-langkah menangani kegagalan. Anda dapat menentukan upaya maksimum, interval backoff, dan kondisi untuk mencoba kembali.

Untuk referensi lengkap tentang helper strategi coba ulang, preset, dan strategi kustom, lihat Mencoba ulang dalam dokumentasi Durable Execution SDK.

Pengecualian di luar langkah

Ketika pengecualian yang tidak tertangkap terjadi dalam kode handler Anda tetapi di luar langkah apa pun, SDK menandai eksekusi sebagai gagal. Ini memastikan kesalahan dalam logika aplikasi Anda ditangkap dan dilaporkan dengan benar.

Skenario	Apa yang terjadi	Dampak pengukuran
Pengecualian dalam kode handler di luar langkah apa pun	SDK menandai eksekusi sebagai GAGAL dan mengembalikan kesalahan . Pengecualian tidak dicoba ulang secara otomatis.	Kesalahan ukuran payload

Untuk mengaktifkan percobaan ulang otomatis untuk kode rawan kesalahan, bungkus dalam satu langkah dengan strategi coba lagi. Langkah-langkah memberikan percobaan ulang otomatis dengan backoff yang dapat dikonfigurasi, sementara kode di luar langkah segera gagal.

Percobaan ulang doa

Percobaan ulang tingkat pemanggilan ditangani secara berbeda tergantung pada bagaimana fungsi tahan lama Lambda dicoba untuk dipanggil. Tabel berikut menjelaskan bagaimana jenis pemanggilan yang berbeda dapat memengaruhi percobaan ulang tingkat pemanggilan.

Jenis doa	Apa yang terjadi
Invokasi sinkron	Lambda tidak secara otomatis mencoba kembali pemanggilan pada kesalahan selama eksekusi fungsi tahan lama. Mencoba lagi pada kegagalan pemanggilan tergantung pada sumber pemanggilan sinkron. Misalnya, menggunakan AWS SDK, InternalFailure dan secara default ThrottlingException dicoba ulang secara otomatis.
Invokasi asinkron	Jika eksekusi fungsi tahan lama gagal (misalnya, ia memasuki status FAILED, STOPTED, atau TIMED_OUT

Jenis doa	Apa yang terjadi
	), Lambda tidak mencoba lagi eksekusi. Ini berbeda dari fungsi tanpa eksekusi yang tahan lama, di mana Lambda mencoba ulang fungsi pada kegagalan pemanggilan asinkron. `MaximumRetryAttempts` Pengaturan untuk pemanggilan asinkron tidak berlaku untuk eksekusi yang tahan lama. Jika Anda mengonfigurasi antrian huruf mati (DLQ) untuk fungsi tersebut, Lambda mengirimkan peristiwa pemicu ke DLQ.
ESM (Pemetaan Sumber Acara)	Lambda secara default mencoba ulang seluruh batch hingga berhasil. Untuk sumber aliran (DynamoDB dan Kinesis), Anda dapat mengonfigurasi jumlah maksimum kali Lambda mencoba ulang saat fungsi Anda mengembalikan kesalahan. Lihat kumpulan pemetaan sumber acara. Untuk Amazon SQS ESM, Anda dapat mengonfigurasi percobaan ulang maksimal melalui DLQ pada antrian Amazon SQS asli. Lihat mengkonfigurasi Amazon SQS ESM. Sebagai praktik terbaik, Anda dapat mempertimbangkan DLQ di tingkat fungsi dan Lambda mengirimkan peristiwa pemicu yang gagal ke DLQ. Lihat fungsi DLQ.

Jenis doa	Apa yang terjadi
Pemicu Langsung	Ini tergantung pada “Pemicu”. Misalnya, Lambda memproses fungsi yang dipicu oleh notifikasi peristiwa Amazon S3 secara asinkron. Lihat Memproses notifikasi acara Amazon SQS dengan Lambda. Lambda memproses fungsi yang dipicu oleh pemberitahuan acara Amazon SNS, secara asinkron. Lihat Memanggil fungsi Lambda dengan notifikasi Amazon SNS. Perilaku coba lagi pemanggilan asinkron ada di atas dalam entri tabel “Pemanggilan asinkron”. Jika Amazon SNS tidak dapat mencapai Lambda atau pesan ditolak, Amazon SNS melakukan percobaan ulang dengan interval yang diperpanjang selama beberapa jam. Untuk perincian nya, lihat Reliabilitas di FAQ Amazon SNS. API Gateway secara sinkron memanggil Lambda dan mengembal ikan respons kesalahan asli kembali ke pemohon. Lihat percobaan ulang doa. Perilaku coba lagi pemanggil an sinkron ada di atas dalam entri tabel “Pemanggi lan sinkron”. Lihat setiap pemicu langsung untuk detail selengkapnya.

Backend mencoba lagi

Percobaan ulang backend terjadi ketika Lambda mengalami kegagalan infrastruktur, kesalahan runtime, atau saat SDK tidak dapat berkomunikasi dengan layanan eksekusi tahan lama. Lambda secara otomatis mencoba kembali kegagalan ini untuk membantu fungsi tahan lama Anda pulih dari masalah infrastruktur sementara.

Skenario coba lagi backend

Lambda secara otomatis mencoba ulang fungsi Anda ketika menemukan skenario berikut:

- Kesalahan layanan internal - Ketika Lambda atau layanan eksekusi tahan lama mengembalikan kesalahan 5xx, menunjukkan masalah layanan sementara.
- Throttling - Ketika fungsi Anda dibatasi karena batas konkurensi atau kuota layanan.

- Timeout - Ketika SDK tidak dapat mencapai layanan eksekusi tahan lama dalam periode batas waktu.
- Kegagalan inisialisasi kotak pasir - Ketika Lambda tidak dapat menginisialisasi lingkungan eksekusi.
- Kesalahan runtime - Ketika runtime Lambda menemukan kesalahan di luar kode fungsi Anda, seperti kesalahan kehabisan memori atau proses crash.
- Kesalahan token pos pemeriksaan tidak valid - Ketika token pos pemeriksaan tidak lagi valid, biasanya karena perubahan status sisi layanan.

Tabel berikut menjelaskan cara SDK menangani skenario ini:

Skenario	Apa yang terjadi	Dampak pengukuran
Kesalahan runtime di luar handler tahan lama (OOM, batas waktu, crash)	Lambda secara otomatis mencoba ulang pemanggilan. SDK diputar ulang dari pos pemeriksaan terakhir, melewatkan langkah-langkah yang telah selesai.	Kesalahan ukuran payload+1 operasi per coba lagi
Kesalahan layanan (5xx) atau batas waktu saat menelepon/API `CheckpointDurableExecution GetDurableExecutionState`	Lambda secara otomatis mencoba ulang pemanggilan. SDK diputar ulang dari pos pemeriksaan terakhir.	Kesalahan ukuran payload+1 operasi per coba lagi
Throttling (429) atau token pos pemeriksaan tidak valid saat memanggil/API `CheckpointDurableExecution GetDurableExecutionState`	Lambda secara otomatis mencoba ulang pemanggilan dengan backoff eksponensial. SDK diputar ulang dari pos pemeriksaan terakhir.	Kesalahan ukuran payload+1 operasi per coba lagi
Kesalahan klien (4xx, kecuali 429 dan token tidak valid) saat/API	SDK menandai eksekusi sebagai GAGAL. Tidak ada percobaan ulang	Kesalahan ukuran payload

Skenario	Apa yang terjadi	Dampak pengukuran
`CheckpointDurableExecution GetDurableExecutionState`	otomatis yang terjadi karena kesalahan menunjukkan masalah permanen.	

Backend mencoba menggunakan backoff eksponensial dan melanjutkan sampai fungsi berhasil atau batas waktu eksekusi tercapai. Selama pemutaran ulang, SDK melewati pos pemeriksaan yang telah selesai dan melanjutkan eksekusi dari operasi terakhir yang berhasil, memastikan fungsi Anda tidak menjalankan kembali pekerjaan yang telah selesai.

Coba lagi praktik terbaik

Ikuti praktik terbaik ini saat mengonfigurasi strategi coba lagi:

- Konfigurasikan strategi coba ulang eksplisit - Jangan mengandalkan perilaku coba ulang default dalam produksi. Konfigurasikan strategi coba ulang eksplisit dengan upaya maksimal dan interval backoff yang sesuai untuk kasus penggunaan Anda.
- Gunakan percobaan ulang bersyarat - Terapkan `shouldRetry` logika untuk mencoba lagi hanya kesalahan sementara (batas tingkat, batas waktu) dan gagal cepat pada kesalahan permanen (kegagalan validasi, tidak ditemukan).
- Tetapkan upaya maksimal yang sesuai - Keseimbangan antara ketahanan dan waktu eksekusi. Terlalu banyak percobaan ulang dapat menunda deteksi kegagalan, sementara terlalu sedikit dapat menyebabkan kegagalan yang tidak perlu.
- Gunakan backoff eksponensial - Backoff eksponensial mengurangi beban pada layanan hilir dan meningkatkan kemungkinan pemulihan dari kegagalan sementara.
- Bungkus kode rawan kesalahan dalam langkah-langkah - Kode di luar langkah tidak dapat dicoba ulang secara otomatis. Bungkus panggilan API eksternal, kueri database, dan operasi rawan kesalahan lainnya dalam langkah-langkah dengan strategi coba lagi.
- Pantau metrik coba lagi - Lacak operasi percobaan ulang langkah dan kegagalan eksekusi di Amazon CloudWatch untuk mengidentifikasi pola dan mengoptimalkan strategi coba lagi.

Idempotensi

Fungsi tahan lama menyediakan idempotensi bawaan untuk eksekusi dimulai melalui nama eksekusi. Saat Anda memberikan nama eksekusi, Lambda menggunakannya untuk mencegah eksekusi

duplikat dan mengaktifkan percobaan ulang permintaan pemanggilan yang aman. Langkah-langkah memiliki semantik eksekusi setidaknya sekali secara default. Selama pemutaran ulang, SDK mengembalikan hasil yang telah diperiksa tanpa mengeksekusi ulang langkah-langkah yang telah selesai, tetapi logika bisnis Anda harus mampu menangani percobaan ulang potensial sebelum selesai.

> **Note**
>
> Pemetaan sumber peristiwa Lambda (ESM) tidak mendukung idempotensi saat peluncuran. Oleh karena itu, setiap pemanggilan (termasuk percobaan ulang) memulai eksekusi baru yang tahan lama. Untuk memastikan eksekusi idempotent dengan pemetaan sumber peristiwa, terapkan logika idempotensi dalam kode fungsi Anda seperti dengan Powertools untuk AWS Lambda atau gunakan fungsi Lambda biasa sebagai proxy (operator) untuk memanggil fungsi tahan lama dengan kunci idempotensi (parameter nama eksekusi).

Nama eksekusi

Anda dapat memberikan nama eksekusi saat memanggil fungsi tahan lama. Nama eksekusi bertindak sebagai kunci idempotensi, memungkinkan Anda untuk mencoba kembali permintaan pemanggilan dengan aman tanpa membuat eksekusi duplikat. Jika Anda tidak memberikan nama, Lambda akan menghasilkan ID eksekusi unik secara otomatis.

Nama eksekusi harus unik dalam akun dan wilayah Anda. Ketika Anda memanggil fungsi dengan nama eksekusi yang sudah ada, perilaku Lambda bergantung pada status eksekusi yang ada dan apakah payload cocok.

Perilaku idempotensi

Tabel berikut menjelaskan cara Lambda menangani permintaan pemanggilan berdasarkan apakah Anda memberikan nama eksekusi, status eksekusi yang ada, dan apakah payload cocok:

Skenario	Nama yang disediakan?	Status eksekusi yang ada	Muatan identik?	Perilaku
1	Tidak	N/A	N/A	Eksekusi baru dimulai: Lambda

Skenario	Nama yang disediakan?	Status eksekusi yang ada	Muatan identik?	Perilaku
				menghasilkan ID eksekusi unik dan memulai eksekusi baru
2	Ya	Tidak pernah ada atau retensi kedaluwarsa	N/A	Eksekusi baru dimulai: Lambda memulai eksekusi baru dengan nama yang disediakan
3	Ya	Berjalan	Ya	Idempotent start: Lambda mengembalikan informasi eksekusi yang ada tanpa memulai duplikat. Untuk pemanggilan sinkron, ini bertindak sebagai lampiran kembali ke eksekusi yang sedang berjalan

Skenario	Nama yang disediakan?	Status eksekusi yang ada	Muatan identik?	Perilaku
4	Ya	Berjalan	Tidak	Kesalahan: Lambda mengembalikan `DurableExecutionAlreadyStartedException` kesalahan karena eksekusi dengan nama ini sudah berjalan dengan payload yang berbeda
5	Ya	Ditutup (berhasil, gagal, berhenti, atau waktu habis)	Ya	Idempotent start: Lambda mengembalikan informasi eksekusi yang ada tanpa memulai eksekusi baru. Hasil eksekusi tertutup dikembalikan
6	Ya	Ditutup (berhasil, gagal, berhenti, atau waktu habis)	Tidak	Kesalahan: Lambda mengembalikan `DurableExecutionAlreadyStartedException` kesalahan karena eksekusi dengan nama ini sudah selesai dengan payload yang berbeda

Catatan

Skenario 3 dan 5 menunjukkan perilaku idempotent di mana Lambda dengan aman menangani permintaan pemanggilan duplikat dengan mengembalikan informasi eksekusi yang ada alih-alih membuat duplikat.

Langkah idempotensi

Langkah-langkah memiliki semantik eksekusi setidaknya sekali secara default. Saat fungsi Anda diputar ulang setelah menunggu, panggilan balik, atau gagal, SDK memeriksa setiap langkah terhadap log pos pemeriksaan. Untuk langkah-langkah yang sudah selesai, SDK mengembalikan hasil checkpoint tanpa mengeksekusi ulang logika langkah. Namun, jika langkah gagal atau fungsi terganggu sebelum langkah selesai, langkah tersebut mungkin dijalankan beberapa kali.

Logika bisnis di dalam langkah Anda harus mampu menangani potensi percobaan ulang. Gunakan kunci idempotensi untuk memastikan operasi seperti pembayaran atau penulisan database hanya dijalankan sekali, bahkan jika langkah tersebut dicoba ulang. Lihat Idempotensi dan percobaan ulang di Panduan Pengembang SDK Ek AWS sekusi Tahan Lama untuk detail tentang cara membuat kode untuk idempotensi.

Menguji fungsi Lambda yang tahan lama

Gunakan SDK pengujian Eksekusi Tahan Lama untuk menjalankan dan memeriksa eksekusi baik secara lokal maupun di cloud. Untuk membuat pengujian, pernyataan, cloud runner, integrasi SAM CLI, dan contoh lengkap, lihat Pengujian di Panduan Pengembang SDK Eksekusi Tahan Lama. AWS

Izin IAM untuk pengujian cloud

Saat Anda menggunakan cloud runner atau `sam remote invoke` untuk menguji fungsi tahan lama yang diterapkan, prinsipal pemanggil memerlukan izin untuk menjalankan fungsi dan membaca riwayat eksekusinya. Lampirkan izin berikut ke penelepon pengujian Anda:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "lambda:InvokeFunction",
```

```
                "lambda:GetDurableExecution",
                "lambda:GetDurableExecutionHistory"
            ],
            "Resource": [
                "arn:aws:lambda:region:account-id:function:function-name",
                "arn:aws:lambda:region:account-id:function:function-name:*"
            ]
        }
    ]
}
```

Ganti *regionaccount-id*,, dan *function-name* dengan nilai-nilai Anda.

Kegagalan debugging

Ketika tes gagal, periksa hasil eksekusi untuk memahami apa yang salah. Periksa status eksekusi untuk melihat apakah fungsi berhasil, gagal, atau habis waktu. Baca pesan kesalahan untuk memahami penyebab kegagalan.

Periksa hasil operasi individu untuk menemukan di mana perilaku menyimpang dari harapan. Periksa hasil langkah untuk melihat nilai apa yang dihasilkan. Verifikasi urutan operasi untuk mengonfirmasi operasi yang dijalankan dalam urutan yang diharapkan. Hitung operasi untuk memastikan jumlah langkah, menunggu, dan panggilan balik yang tepat dibuat.

Masalah umum termasuk kode non-deterministik yang menghasilkan hasil yang berbeda pada pemutaran ulang, status bersama melalui variabel global yang rusak selama pemutaran ulang, dan operasi yang hilang karena kesalahan logika bersyarat. Gunakan debugger standar dan logging untuk melangkah melalui kode fungsi dan melacak alur eksekusi.

Untuk pengujian cloud, periksa riwayat eksekusi di CloudWatch Log untuk melihat log operasi terperinci. Gunakan tracing untuk melacak alur eksekusi di seluruh layanan dan mengidentifikasi kemacetan.

Memantau fungsi tahan lama

Anda dapat memantau fungsi tahan lama Anda menggunakan CloudWatch metrik, CloudWatch Log, dan penelusuran. Karena fungsi tahan lama dapat berjalan untuk waktu yang lama dan menjangkau beberapa pemanggilan fungsi, pemantauannya memerlukan pemahaman pola eksekusi uniknya, termasuk pos pemeriksaan, transisi status, dan perilaku replay.

> **Note**
>
> Untuk informasi selengkapnya tentang kuota yang berlaku untuk fungsi tahan lama, lihat Kuota fungsi tahan lama.

CloudWatch metrik

Lambda secara otomatis menerbitkan metrik ke CloudWatch tanpa biaya tambahan. Fungsi tahan lama menyediakan metrik tambahan di luar metrik Lambda standar untuk membantu Anda memantau alur kerja yang berjalan lama, manajemen status, dan pemanfaatan sumber daya.

Metrik eksekusi yang tahan lama

Lambda memancarkan metrik berikut untuk eksekusi tahan lama:

Metrik	Deskripsi
`ApproximateRunningDurableExecutions`	Jumlah eksekusi tahan lama dalam status RUNNING
`ApproximateRunningDurableExecutionsUtilization`	Persentase kuota eksekusi tahan lama maksimum akun Anda yang saat ini digunakan
`DurableExecutionDuration`	Waktu jam dinding yang berlalu dalam milidetik sehingga eksekusi yang tahan lama tetap dalam keadaan RUNNING
`DurableExecutionStarted`	Jumlah eksekusi tahan lama yang dimulai
`DurableExecutionStopped`	Jumlah eksekusi tahan lama yang dihentikan menggunakan StopDurableExecution API
`DurableExecutionSucceeded`	Jumlah eksekusi tahan lama yang berhasil diselesaikan
`DurableExecutionFailed`	Jumlah eksekusi tahan lama yang diselesaikan dengan kegagalan

Metrik	Deskripsi
`DurableExecutionTimedOut`	Jumlah eksekusi tahan lama yang melebihi batas waktu eksekusi yang dikonfigurasi
`DurableExecutionOperations`	Jumlah operasi kumulatif yang dilakukan dalam eksekusi yang tahan lama
`DurableExecutionStorageWrittenBytes`	Jumlah kumulatif data dalam byte dipertahankan oleh eksekusi yang tahan lama

CloudWatch metrik

Lambda memancarkan metrik pemanggilan, kinerja, dan konkurensi standar untuk fungsi yang tahan lama. Karena eksekusi yang tahan lama dapat mencakup beberapa pemanggilan fungsi saat berlangsung melalui pos pemeriksaan dan pemutaran ulang, metrik ini berperilaku berbeda dari fungsi standar:

- Pemanggilan: Menghitung setiap pemanggilan fungsi, termasuk pemutaran ulang. Eksekusi tahan lama tunggal dapat menghasilkan beberapa titik data pemanggilan.
- Durasi: Mengukur setiap pemanggilan fungsi secara terpisah. Gunakan `DurableExecutionDuration` untuk total waktu yang dibutuhkan oleh satu eksekusi tahan lama.
- Kesalahan: Melacak kegagalan pemanggilan fungsi. Gunakan `DurableExecutionFailed` untuk kegagalan tingkat eksekusi.

Untuk daftar lengkap metrik Lambda standar, lihat Jenis metrik untuk fungsi Lambda.

Menciptakan CloudWatch alarm

Buat CloudWatch alarm untuk memberi tahu Anda saat metrik melebihi ambang batas. Alarm umum meliputi:

- `ApproximateRunningDurableExecutionsUtilization` melebihi 80% dari kuota Anda
- `DurableExecutionFailed` meningkat di atas ambang batas
- `DurableExecutionTimedOut` menunjukkan waktu eksekusi habis
- `DurableExecutionStorageWrittenBytes` mendekati batas penyimpanan

Untuk informasi lebih lanjut, lihat Menggunakan CloudWatch alarm.

EventBridge acara

Lambda menerbitkan peristiwa perubahan status eksekusi tahan lama ke EventBridge. Anda dapat menggunakan peristiwa ini untuk memicu alur kerja, mengirim pemberitahuan, atau melacak perubahan siklus hidup eksekusi di seluruh fungsi tahan lama Anda.

Peristiwa perubahan status eksekusi yang tahan lama

Lambda memancarkan peristiwa ke EventBridge setiap kali eksekusi yang tahan lama berubah status. Peristiwa ini memiliki karakteristik sebagai berikut:

- Sumber: `aws.lambda`
- Tipe detail: `Durable Execution Status Change`

Peristiwa perubahan status dipublikasikan untuk status eksekusi berikut:

- `RUNNING` - Eksekusi dimulai
- `SUCCEEDED` - Eksekusi selesai dengan sukses
- `STOPPED` - Eksekusi berhenti menggunakan StopDurableExecution API
- `FAILED` - Eksekusi gagal dengan kesalahan
- `TIMED_OUT` - Eksekusi melebihi batas waktu yang dikonfigurasi

Contoh berikut menunjukkan peristiwa perubahan status eksekusi yang tahan lama:

```
{
  "version": "0",
  "id": "d019b03c-a8a3-9d58-85de-241e96206538",
  "detail-type": "Durable Execution Status Change",
  "source": "aws.lambda",
  "account": "123456789012",
  "time": "2025-11-20T13:08:22Z",
  "region": "us-east-1",
  "resources": [],
  "detail": {
    "durableExecutionArn": "arn:aws:lambda:us-east-1:123456789012:function:my-function:
$LATEST/durable-execution/090c4189-b18b-4296-9d0c-cfd01dc3a122/9f7d84c9-ea3d-3ffc-
b3e5-5ec51c34ffc9",
```

```
    "durableExecutionName": "order-123",
    "functionArn": "arn:aws:lambda:us-east-1:123456789012:function:my-function:2",
    "status": "RUNNING",
    "startTimestamp": "2025-11-20T13:08:22.345Z"
  }
}
```

Untuk status terminal (`SUCCEEDED`, `STOPPED`, `FAILED`, `TIMED_OUT`), acara tersebut meliputi `endTimestamp` bidang yang menunjukkan kapan eksekusi selesai.

Menciptakan EventBridge aturan

Buat EventBridge aturan untuk merutekan peristiwa perubahan status eksekusi tahan lama ke target seperti Amazon Simple Notification Service, Amazon Simple Queue Service, atau fungsi Lambda lainnya.

Contoh berikut membuat aturan yang cocok dengan semua perubahan status eksekusi tahan lama:

```
{
  "source": ["aws.lambda"],
  "detail-type": ["Durable Execution Status Change"]
}
```

Contoh berikut membuat aturan yang hanya cocok dengan eksekusi yang gagal:

```
{
  "source": ["aws.lambda"],
  "detail-type": ["Durable Execution Status Change"],
  "detail": {
    "status": ["FAILED"]
  }
}
```

Contoh berikut membuat aturan yang cocok dengan perubahan status untuk fungsi tertentu:

```
{
```

```
  "source": ["aws.lambda"],
  "detail-type": ["Durable Execution Status Change"],
  "detail": {
    "functionArn": [{
      "prefix": "arn:aws:lambda:us-east-1:123456789012:function:my-function"
    }]
  }
}
```

Untuk informasi lebih lanjut tentang membuat EventBridge aturan, lihat Amazon EventBridge tutorial di EventBridge Panduan Pengguna.

Menggunakan antrian huruf mati dengan EventBridge acara

EventBridge peristiwa perubahan status memberikan pemberitahuan waktu nyata ketika eksekusi tahan lama memasuki status FAILED, STOPED, atau TIMED_OUT, tetapi tidak menyertakan payload peristiwa asli yang memicu eksekusi. Untuk menangkap peristiwa pemicu untuk analisis atau pemrosesan ulang nanti, konfigurasikan antrean huruf mati (DLQ) pada fungsi tahan lama Anda.

Ketika eksekusi tahan lama gagal setelah pemanggilan asinkron, Lambda mengirimkan peristiwa pemicu asli ke DLQ yang dikonfigurasi. Gunakan EventBridge acara dan DLQ bersama-sama untuk strategi penanganan kegagalan yang komprehensif:

- EventBridge acara — Memicu peringatan segera, perbarui dasbor, atau mulai alur kerja remediasi saat eksekusi gagal.
- Dead-letter mengantre — Pertahankan payload acara asli sehingga Anda dapat memeriksa apa yang menyebabkan kegagalan, men-debug masalah, dan secara opsional memproses ulang acara setelah memperbaiki masalah yang mendasarinya.

Untuk informasi tentang mengkonfigurasi DLQ pada fungsi Anda, lihat Dead-letter mengantre.

AWS X-Ray melacak

Anda dapat mengaktifkan X-Ray menelusuri fungsi tahan lama Anda. Lambda melewati X-Ray melacak header ke eksekusi tahan lama, memungkinkan Anda untuk melacak permintaan di seluruh alur kerja Anda.

Untuk mengaktifkan X-Ray; pelacakan menggunakan konsol Lambda, pilih fungsi Anda, lalu pilih Alat Konfigurasi, Pemantauan dan operasi, dan aktifkan Pelacakan aktif di bawah X-Ray.

Untuk mengaktifkan X-Ray menelusuri menggunakan AWS CLI:

```
aws lambda update-function-configuration \
    --function-name my-durable-function \
    --tracing-config Mode=Active
```

Untuk mengaktifkan AWS X-Ray menelusuri menggunakan AWS SAM:

```
Resources:
  MyDurableFunction:
    Type: AWS::Serverless::Function
    Properties:
      Tracing: Active
      DurableConfig:
        ExecutionTimeout: 3600
```

Untuk informasi lebih lanjut tentang X-Ray, lihat AWS X-Ray Panduan Pengembang.

Praktik terbaik untuk fungsi tahan lama Lambda

Fungsi tahan lama menggunakan model eksekusi berbasis replay yang membutuhkan praktik pemrograman yang berbeda dari fungsi Lambda tradisional. Lihat Praktik terbaik di Panduan Pengembang SDK Eksekusi AWS Tahan Lama untuk panduan tentang cara menulis dan menguji kode alur kerja yang tahan lama.

Rekomendasi berikut adalah praktik terbaik untuk menerapkan, menjalankan, dan memantau fungsi tahan lama Lambda.

Versi fungsi dan alias

Memanggil fungsi dengan nomor versi atau alias untuk menyematkan eksekusi ke versi kode tertentu. Pastikan versi kode baru dapat menangani status dari versi yang lebih lama. Jangan mengganti nama langkah atau mengubah perilaku mereka dengan cara yang merusak pemutaran ulang. Jika Anda mengganti nama langkah atau mengubah perilakunya saat eksekusi sedang berlangsung, eksekusi tersebut dapat gagal untuk melanjutkan atau menghasilkan hasil yang salah. Ini terjadi karena runtime tidak dapat lagi mencocokkan status tersimpan dengan definisi langkah yang diharapkan.

Memantau

Aktifkan logging terstruktur dengan ID eksekusi dan nama langkah. Siapkan CloudWatch alarm untuk tingkat kesalahan dan durasi eksekusi. Gunakan penelusuran untuk mengidentifikasi kemacetan. Untuk panduan terperinci, lihat Monitoring dan debugging.

Penanganan kesalahan

Selain mengonfigurasi strategi coba ulang untuk kegagalan sementara, konfigurasikan antrian huruf mati (DLQ) pada fungsi tahan lama Anda untuk menangkap peristiwa dari eksekusi yang gagal secara permanen. Ketika eksekusi tahan lama mencapai status terminal (GAGAL, BERHENTI, atau TIMED_OUT) setelah pemanggilan asinkron, Lambda mengirimkan peristiwa pemicu asli ke DLQ. Dengan cara ini, Anda dapat memeriksa, men-debug, dan secara opsional memproses ulang peristiwa yang gagal tanpa kehilangannya.

Untuk mengonfigurasi DLQ, setel `DeadLetterConfig` properti pada fungsi Anda ke antrean Amazon SQS atau ARN topik Amazon SNS. Untuk informasi lebih lanjut, lihat Dead-letter antrian.

Ikuti praktik terbaik berikut untuk penanganan kesalahan dengan fungsi tahan lama:

- Konfigurasikan DLQ untuk pemanggilan asinkron — Selalu lampirkan antrian huruf mati saat menjalankan fungsi tahan lama secara asinkron. Tidak seperti pemanggilan fungsi asinkron tanpa eksekusi yang tahan lama, Lambda tidak secara otomatis mencoba kembali eksekusi yang tahan lama pada kegagalan. DLQ yang dikonfigurasi dengan benar menangkap peristiwa yang menyebabkan eksekusi gagal secara permanen. Pastikan DLQ memiliki izin yang benar dan kapasitas yang cukup untuk menerima pesan.
- Gunakan strategi coba lagi dalam langkah-langkah — Konfigurasikan strategi coba ulang eksplisit dengan backoff yang sesuai untuk kegagalan sementara. Untuk panduan tentang mengonfigurasi percobaan ulang, lihat Mencoba ulang untuk fungsi yang tahan lama.
- Gabungkan DLQ dengan EventBridge notifikasi — Gunakan EventBridge aturan untuk memperingatkan perubahan status GAGAL, BERHENTI, dan TIMED_OUT untuk visibilitas waktu nyata, dan gunakan DLQ untuk mempertahankan muatan peristiwa asli untuk analisis atau pemrosesan ulang nanti.
- Pantau kedalaman DLQ — Buat CloudWatch alarm pada `ApproximateNumberOfMessagesVisible` metrik agar DLQ Anda dapat mendeteksi saat kegagalan menumpuk.

Sumber daya terkait

- AWS Panduan Pengembang SDK Eksekusi Tahan Lama
- Memantau fungsi Lambda yang tahan lama
- Mencoba lagi untuk fungsi tahan lama Lambda
- Menguji fungsi Lambda yang tahan lama

Meningkatkan kinerja startup dengan Lambda SnapStart

Lambda SnapStart dapat memberikan kinerja startup serendah sub-detik, biasanya tanpa perubahan pada kode Fungsi Acara Anda. SnapStart membuatnya lebih mudah untuk membangun aplikasi yang sangat responsif dan skalabel tanpa menyediakan sumber daya atau menerapkan pengoptimalan kinerja yang kompleks.

Kontributor terbesar untuk latensi startup (sering disebut sebagai cold start time) adalah waktu yang dihabiskan Lambda untuk menginisialisasi fungsi, yang mencakup memuat kode fungsi, memulai runtime, dan menginisialisasi kode fungsi. Dengan SnapStart, Lambda menginisialisasi Fungsi Acara Anda saat Anda mempublikasikan versi fungsi. Lambda mengambil snapshot MicroVM Firecracker dari memori dan status disk dari lingkungan eksekusi yang diinisialisasi, mengenkripsi snapshot, dan menyimpannya secara cerdas untuk mengoptimalkan latensi pengambilan.

Untuk memastikan ketahanan, Lambda menyimpan beberapa salinan dari setiap snapshot. Lambda secara otomatis menambal snapshot dan salinannya dengan runtime terbaru dan pembaruan keamanan. Saat Anda memanggil versi Fungsi Acara untuk pertama kalinya, dan saat pemanggilan meningkat, Lambda melanjutkan lingkungan eksekusi baru dari snapshot yang di-cache alih-alih menginisialisasinya dari awal, meningkatkan latensi startup.

> **Important**
>
> Jika aplikasi Anda bergantung pada keunikan status, Anda harus mengevaluasi kode fungsi Anda dan memverifikasi bahwa itu tahan terhadap operasi snapshot. Untuk informasi selengkapnya, lihat Menangani keunikan dengan Lambda SnapStart.

Jika Anda mengalami masalah, lihatthe section called “Pemecahan masalah”.

Topik

- Kapan harus menggunakan SnapStart
- Fitur yang didukung dan batasan
- Wilayah yang Didukung
- Pertimbangan kompatibilitas
- SnapStart harga
- Mengaktifkan dan mengelola Lambda SnapStart
- Menangani keunikan dengan Lambda SnapStart
- Menerapkan kode sebelum atau sesudah snapshot Fungsi Peristiwa Lambda
- Pemantauan untuk Lambda SnapStart
- Model keamanan untuk Lambda SnapStart
- Maksimalkan kinerja Lambda SnapStart
- Memecahkan masalah SnapStart kesalahan untuk Fungsi Peristiwa Lambda

Kapan harus menggunakan SnapStart

Lambda SnapStart dirancang untuk mengatasi variabilitas latensi yang diperkenalkan oleh kode inisialisasi satu kali, seperti memuat dependensi modul atau kerangka kerja. Operasi ini terkadang membutuhkan waktu beberapa detik untuk diselesaikan selama pemanggilan awal. Gunakan SnapStart untuk mengurangi latensi ini dari beberapa detik menjadi serendah sub-detik, dalam skenario optimal. SnapStart bekerja paling baik saat digunakan dengan pemanggilan fungsi dalam skala besar. Fungsi yang jarang dipanggil mungkin tidak mengalami peningkatan kinerja yang sama.

SnapStart sangat bermanfaat untuk dua jenis aplikasi utama:

- Latency-sensitive API dan alur pengguna: Fungsi yang merupakan bagian dari titik akhir API kritis atau alur yang dihadapi pengguna dapat memperoleh manfaat dari SnapStart pengurangan latensi dan waktu respons yang ditingkatkan.
- Latency-sensitive alur kerja pemrosesan data: alur kerja pemrosesan Time-bound data yang menggunakan Lambda Event Functions dapat mencapai throughput yang lebih baik dengan mengurangi latensi inisialisasi fungsi outlier.

Konkurensi yang disediakan membuat Fungsi Acara diinisialisasi dan siap merespons dalam milidetik dua digit. Gunakan konkurensi yang disediakan jika aplikasi Anda memiliki persyaratan latensi start dingin yang ketat yang tidak dapat ditangani secara memadai oleh. SnapStart

Fitur yang didukung dan batasan

SnapStart mendukung runtime terkelola Lambda berikut dan gambar dasar AWS Lambda yang sesuai di seluruh format penyebaran gambar ZIP dan kontainer:

- Java 11 dan setelahnya
- Python 3.12 dan yang lebih baru
- .NET 8 dan setelahnya. Jika Anda menggunakan kerangka kerja Anotasi Lambda untuk.NET, tingkatkan ke Amazon.Lambda.Annotationsversi 1.6.0 atau yang lebih baru untuk memastikan kompatibilitas dengan. SnapStart

Untuk gambar kontainer yang dibangun di atas gambar dasar AWS Lambda atau gambar dasar kustom, lihat Menerapkan SnapStart kait untuk gambar kontainer dan validasi kode fungsi Anda untuk menangani SnapStart persyaratan keunikan.

Runtime terkelola lainnya (seperti `nodejs24.x` dan`ruby4.0`) dan tidak OS-only runtime didukung. SnapStart tidak mendukung konkurensi yang disediakan, Amazon Elastic File System (Amazon EFS), File Amazon S3, atau penyimpanan sementara yang lebih besar dari 512 MB.

> **Note**
>
> Anda SnapStart hanya dapat menggunakan pada versi fungsi yang diterbitkan dan alias yang mengarah ke versi. Anda tidak dapat menggunakan SnapStart pada versi fungsi yang tidak dipublikasikan ($LATEST).

Wilayah yang Didukung

Lambda SnapStart tersedia di semua Wilayah komersial kecuali Asia Pasifik (Selandia Baru) dan Asia Pasifik (Taipei).

Pertimbangan kompatibilitas

Dengan SnapStart, Lambda menggunakan satu snapshot sebagai status awal untuk beberapa lingkungan eksekusi. Jika fungsi Anda menggunakan salah satu dari berikut ini selama fase inisialisasi, maka Anda mungkin perlu membuat beberapa perubahan sebelum menggunakan SnapStart:

Keunikan

Jika kode inisialisasi Anda menghasilkan konten unik yang disertakan dalam snapshot, maka konten tersebut mungkin tidak unik saat digunakan kembali di seluruh lingkungan eksekusi. Untuk mempertahankan keunikan saat menggunakan SnapStart, Anda harus menghasilkan konten unik setelah inisialisasi. Ini termasuk ID unik, rahasia unik, dan entropi yang digunakan untuk menghasilkan pseudorandomness. Untuk mempelajari cara mengembalikan keunikan, lihat. Menangani keunikan dengan Lambda SnapStart

Koneksi jaringan

Status koneksi yang dibuat oleh Event Function Anda selama fase inisialisasi tidak dijamin saat Lambda melanjutkan fungsi Anda dari snapshot. Validasi status koneksi jaringan Anda dan buat kembali seperlunya. Dalam kebanyakan kasus, koneksi jaringan yang dibuat AWS SDK secara otomatis dilanjutkan. Untuk koneksi lain, tinjau praktik terbaik.

Data sementara

Beberapa fungsi mengunduh atau menginisialisasi data sementara, seperti kredensil sementara atau stempel waktu yang di-cache, selama fase inisialisasi. Segarkan data singkat di penangan fungsi sebelum menggunakannya, bahkan saat tidak menggunakan. SnapStart

SnapStart harga

> **Note**
>
> Untuk runtime terkelola Java, tidak ada biaya tambahan untuk SnapStart. Anda dikenakan biaya berdasarkan jumlah permintaan untuk fungsi Anda, waktu yang dibutuhkan kode Anda untuk menjalankan, dan memori yang dikonfigurasi untuk fungsi Anda.

Biaya penggunaan SnapStart meliputi:

- Caching: Untuk setiap versi fungsi yang Anda publikasikan dengan SnapStart diaktifkan, Anda membayar biaya caching dan mempertahankan snapshot. Harganya tergantung pada jumlah memori yang Anda alokasikan untuk fungsi Anda. Anda dikenakan biaya minimal 3 jam.

  Anda terus ditagih selama fungsi Anda tetap aktif. Gunakan tindakan ListVersionsByFunctionAPI untuk mengidentifikasi versi fungsi, lalu gunakan DeleteFunctionuntuk menghapus versi yang

tidak digunakan. Untuk menghapus versi fungsi yang tidak digunakan secara otomatis, lihat pola Pembersihan Versi Lambda di Tanah Tanpa Server.

- Restorasi: Setiap kali instance fungsi dipulihkan dari snapshot, Anda membayar biaya pemulihan. Harganya tergantung pada jumlah memori yang Anda alokasikan ke fungsi Anda.

Seperti semua Fungsi Peristiwa Lambda, biaya durasi berlaku untuk kode yang berjalan di penangan fungsi. Untuk SnapStart fungsi, biaya durasi juga berlaku untuk kode inisialisasi yang dideklarasikan di luar handler, waktu yang diperlukan untuk memuat runtime, dan kode apa pun yang berjalan di hook runtime. Durasi dihitung dari waktu kode Anda mulai berjalan hingga kembali atau berakhir, dibulatkan ke 1 ms terdekat. Lambda menyimpan salinan snapshot Anda yang di-cache untuk ketahanan dan secara otomatis menerapkan pembaruan perangkat lunak, seperti peningkatan runtime dan patch keamanan padanya. Biaya berlaku setiap kali Lambda menjalankan kembali kode inisialisasi Anda untuk menerapkan pembaruan perangkat lunak.

Untuk informasi selengkapnya tentang biaya penggunaan SnapStart, lihat AWS Lambda Harga.

Mengaktifkan dan mengelola Lambda SnapStart

Untuk menggunakan SnapStart, aktifkan SnapStart pada Fungsi Acara Lambda baru atau yang sudah ada. Kemudian, publikasikan dan panggil versi fungsi.

Topik

- Mengaktifkan SnapStart (konsol)
- Mengaktifkan SnapStart (AWS CLI)
- Mengaktifkan SnapStart (API)
- Status Lambda SnapStart dan Fungsi Acara
- Memperbarui snapshot
- Menggunakan SnapStart dengan AWS SDK
- Menggunakan SnapStart dengan CloudFormation, AWS SAM, dan AWS CDK
- Menghapus snapshot

Mengaktifkan SnapStart (konsol)

SnapStart Untuk mengaktifkan Fungsi Acara

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih nama Fungsi Acara.
3. Pilih Konfigurasi, lalu pilih Konfigurasi umum.
4. Pada panel konfigurasi Umum, pilih Edit.
5. Pada halaman Edit pengaturan dasar, untuk SnapStart, pilih Versi yang diterbitkan.
6. Pilih Simpan.
7. Publikasikan versi fungsi. Lambda menginisialisasi kode Anda, membuat snapshot dari lingkungan eksekusi yang diinisialisasi, dan kemudian menyimpan snapshot untuk akses latensi rendah.
8. Memanggil versi fungsi.

Mengaktifkan SnapStart (AWS CLI)

SnapStart Untuk mengaktifkan Fungsi Event yang ada

1. Perbarui konfigurasi fungsi dengan menjalankan perintah update-function-configuration dengan opsi. --snap-start

```
aws lambda update-function-configuration \
  --function-name my-function \
  --snap-start ApplyOn=PublishedVersions
```

2. Publikasikan versi fungsi dengan perintah publish-version.

```
aws lambda publish-version \
  --function-name my-function
```

3. Konfirmasikan bahwa SnapStart diaktifkan untuk versi fungsi dengan menjalankan perintah get-function-configuration dan menentukan nomor versi. Contoh berikut menentukan versi 1.

```
aws lambda get-function-configuration \
  --function-name my-function:1
```

   Jika respons menunjukkan bahwa OptimizationStatusadalah 0n dan State is`Active`, maka SnapStart diaktifkan dan snapshot tersedia untuk versi fungsi yang ditentukan.

```
"SnapStart": {
    "ApplyOn": "PublishedVersions",
    "OptimizationStatus": "On"
 },
 "State": "Active",
```

4. Panggil versi fungsi dengan menjalankan perintah pemanggilan dan menentukan versinya. Contoh berikut memanggil versi 1.

```
aws lambda invoke \
  --cli-binary-format raw-in-base64-out \
  --function-name my-function:1 \
  --payload '{ "name": "Bob" }' \
  response.json
```

cli-binary-formatOpsi ini diperlukan jika Anda menggunakan AWS CLI versi 2. Untuk menjadikan ini pengaturan default, jalankan`aws configure set cli-binary-format raw-in-base64-out`. Untuk informasi selengkapnya, lihat opsi baris perintah global yang AWS CLI didukung di Panduan AWS Command Line Interface Pengguna untuk Versi 2.

Untuk mengaktifkan SnapStart saat Anda membuat Fungsi Acara baru

1. Buat fungsi dengan menjalankan perintah create-function dengan opsi. --snap-start Untuk--role, tentukan Nama Sumber Daya Amazon (ARN) peran eksekusi Anda.

> **Note**
>
> Jika Anda menggunakan SnapStart dengan fungsi gambar kontainer yang menggunakan gambar dasar selain gambar dasar Lambda untuk Java (versi 11+), Python (versi 3.12+), dan .NET (versi 8+), ikuti petunjuk di Menerapkan SnapStart kait untuk gambar kontainer untuk membuat gambar Anda kompatibel dengan. AWS SnapStart

```
aws lambda create-function \
  --function-name my-function \
  --runtime "java25" \
  --zip-file fileb://my-function.zip \
  --handler my-function.handler \
  --role arn:aws:iam::111122223333:role/lambda-ex \
  --snap-start ApplyOn=PublishedVersions
```

2. Buat versi dengan perintah publish-version.

```
aws lambda publish-version \
  --function-name my-function
```

3. Konfirmasikan bahwa SnapStart diaktifkan untuk versi fungsi dengan menjalankan perintah get-function-configuration dan menentukan nomor versi. Contoh berikut menentukan versi 1.

```
aws lambda get-function-configuration \
  --function-name my-function:1
```

Jika respons menunjukkan bahwa OptimizationStatusadalah 0n dan State is`Active`, maka SnapStart diaktifkan dan snapshot tersedia untuk versi fungsi yang ditentukan.

```
"SnapStart": {
     "ApplyOn": "PublishedVersions",
     "OptimizationStatus": "On"
  },
  "State": "Active",
```

4. Panggil versi fungsi dengan menjalankan perintah pemanggilan dan menentukan versinya. Contoh berikut memanggil versi 1.

```
aws lambda invoke \
  --cli-binary-format raw-in-base64-out \
  --function-name my-function:1 \
  --payload '{ "name": "Bob" }' \
  response.json
```

cli-binary-formatOpsi ini diperlukan jika Anda menggunakan AWS CLI versi 2. Untuk menjadikan ini pengaturan default, jalankan`aws configure set cli-binary-format raw-in-base64-out`. Untuk informasi selengkapnya, lihat opsi baris perintah global yang AWS CLI didukung di Panduan AWS Command Line Interface Pengguna untuk Versi 2.

Mengaktifkan SnapStart (API)

Untuk mengaktifkan SnapStart

1. Lakukan salah satu tindakan berikut:
   - Buat Fungsi Acara baru dengan SnapStart diaktifkan dengan menggunakan aksi CreateFunctionAPI dengan SnapStartparameter.
   - Aktifkan SnapStart untuk Fungsi Acara yang ada dengan menggunakan UpdateFunctionConfigurationtindakan dengan SnapStartparameter.
2. Publikasikan versi fungsi dengan PublishVersiontindakan. Lambda menginisialisasi kode Anda, membuat snapshot dari lingkungan eksekusi yang diinisialisasi, dan kemudian menyimpan snapshot untuk akses latensi rendah.
3. Konfirmasikan bahwa SnapStart diaktifkan untuk versi fungsi dengan menggunakan GetFunctionConfigurationtindakan. Tentukan nomor versi untuk mengonfirmasi bahwa SnapStart

diaktifkan untuk versi tersebut. Jika respons menunjukkan bahwa OptimizationStatusadalah 0n dan State isActive, maka SnapStart diaktifkan dan snapshot tersedia untuk versi fungsi yang ditentukan.

```
"SnapStart": {
        "ApplyOn": "PublishedVersions",
        "OptimizationStatus": "On"
    },
    "State": "Active",
```

4. Panggil versi fungsi dengan tindakan Invoke.

Status Lambda SnapStart dan Fungsi Acara

Status fungsi berikut dapat terjadi saat Anda menggunakan SnapStart.

Tertunda

Lambda menginisialisasi kode Anda dan mengambil snapshot dari lingkungan eksekusi yang diinisialisasi. Setiap pemanggilan atau tindakan API lainnya yang beroperasi pada versi fungsi gagal.

Aktif

Pembuatan snapshot selesai dan Anda dapat menjalankan fungsinya. Untuk menggunakannya SnapStart, Anda harus memanggil versi fungsi yang diterbitkan, bukan versi yang tidak dipublikasikan ($LATEST).

Tidak aktif

`Inactive`Status dapat terjadi ketika Lambda secara berkala meregenerasi snapshot fungsi untuk menerapkan pembaruan perangkat lunak. Dalam hal ini, jika fungsi Anda gagal untuk menginisialisasi, fungsi dapat memasukkan `Inactive` status.

Untuk fungsi yang menggunakan runtime Java, Lambda menghapus snapshot setelah 14 hari tanpa pemanggilan. Jika Anda memanggil versi fungsi setelah 14 hari, Lambda mengembalikan `SnapStartNotReadyException` respons dan mulai menginisialisasi snapshot baru. Tunggu hingga versi fungsi mencapai `Active` status, lalu panggil lagi.

Gagal

Lambda mengalami kesalahan saat menjalankan kode inisialisasi atau membuat snapshot.

Memperbarui snapshot

Lambda membuat snapshot untuk setiap versi fungsi yang diterbitkan. Untuk memperbarui snapshot, publikasikan versi fungsi baru.

Menggunakan SnapStart dengan AWS SDK

Untuk membuat panggilan AWS SDK dari fungsi Anda, Lambda menghasilkan kumpulan kredensional sementara dengan mengasumsikan peran eksekusi fungsi Anda. Kredensial ini tersedia sebagai variabel lingkungan selama pemanggilan fungsi Anda. Anda tidak perlu memberikan kredensi untuk SDK secara langsung dalam kode. Secara default, rantai penyedia kredensi secara berurutan memeriksa setiap tempat di mana Anda dapat menyetel kredensialnya dan memilih yang pertama tersedia—biasanya variabel lingkungan (,, dan). `AWS_ACCESS_KEY_ID` `AWS_SECRET_ACCESS_KEY` `AWS_SESSION_TOKEN`

> **Note**
>
> Saat SnapStart diaktifkan, runtime Lambda secara otomatis menggunakan kredensial kontainer (`AWS_CONTAINER_CREDENTIALS_FULL_URI`dan`AWS_CONTAINER_AUTHORIZATION_TOKEN`) alih-alih variabel lingkungan kunci akses. Ini mencegah kredensi kedaluwarsa sebelum fungsi dipulihkan.

Menggunakan SnapStart dengan CloudFormation, AWS SAM, dan AWS CDK

- AWS CloudFormation: Deklarasikan SnapStartentitas dalam template Anda.
- AWS Serverless Application Model (AWS SAM): Deklarasikan SnapStartproperti di template Anda.
- AWS Cloud Development Kit (AWS CDK): Gunakan SnapStartPropertytipenya.

Menghapus snapshot

Lambda menghapus snapshot saat:

- Anda menghapus fungsi atau versi fungsi.
- Hanya runtime Java - Anda tidak memanggil versi fungsi selama 14 hari. Setelah 14 hari tanpa pemanggilan, versi fungsi bertransisi ke status Tidak Aktif. Jika Anda memanggil versi fungsi

setelah 14 hari, Lambda mengembalikan `SnapStartNotReadyException` respons dan mulai menginisialisasi snapshot baru. Tunggu hingga versi fungsi mencapai status Aktif, lalu panggil lagi.

Lambda menghapus semua sumber daya yang terkait dengan snapshot yang dihapus sesuai dengan Peraturan Perlindungan Data Umum (GDPR).

Menangani keunikan dengan Lambda SnapStart

Saat pemanggilan ditingkatkan pada suatu SnapStart fungsi, Lambda menggunakan satu snapshot yang diinisialisasi untuk melanjutkan beberapa lingkungan eksekusi. Jika kode inisialisasi Anda menghasilkan konten unik yang disertakan dalam snapshot, maka konten tersebut mungkin tidak unik saat digunakan kembali di seluruh lingkungan eksekusi. Untuk mempertahankan keunikan saat menggunakan SnapStart, Anda harus menghasilkan konten unik setelah inisialisasi. Ini termasuk ID unik, rahasia unik, dan entropi yang digunakan untuk menghasilkan pseudorandomness.

Kami merekomendasikan praktik terbaik berikut untuk membantu Anda mempertahankan keunikan dalam kode Anda. Untuk fungsi Java, Lambda juga menyediakan alat SnapStart pemindaian sumber terbuka untuk membantu memeriksa kode yang mengasumsikan keunikan. Jika Anda menghasilkan data unik selama fase inisialisasi, maka Anda dapat menggunakan kait runtime untuk mengembalikan keunikan. Dengan runtime hooks, Anda dapat menjalankan kode tertentu segera sebelum Lambda mengambil snapshot atau segera setelah Lambda melanjutkan fungsi dari snapshot.

Hindari menyimpan status yang bergantung pada keunikan selama inisialisasi

Selama fase inisialisasi Fungsi Acara Anda, hindari caching data yang dimaksudkan untuk menjadi unik, seperti membuat ID unik untuk pencatatan atau pengaturan benih untuk fungsi acak. Sebagai gantinya, kami menyarankan Anda membuat data unik atau menetapkan seed untuk fungsi acak di dalam penangan fungsi Anda—atau menggunakan hook runtime.

Contoh berikut menunjukkan cara menghasilkan UUID di handler Event Function.

Java

Example— Menghasilkan ID unik di penangan Fungsi Acara

```
import java.util.UUID;
  public class Handler implements RequestHandler<String, String> {
    private static UUID uniqueSandboxId = null;
    @Override
    public String handleRequest(String event, Context context) {
      if (uniqueSandboxId == null)
        uniqueSandboxId = UUID.randomUUID();
      System.out.println("Unique Sandbox Id: " + uniqueSandboxId);
      return "Hello, World!";
    }
  }
```

Python

Example— Menghasilkan ID unik di penangan Fungsi Acara

```
import json
import random
import time

unique_number = None

def lambda_handler(event, context):
    seed = int(time.time() * 1000)
    random.seed(seed)
    global unique_number
    if not unique_number:
        unique_number = random.randint(1, 10000)

    print("Unique number: ", unique_number)

    return "Hello, World!"
```

.NET

Example— Menghasilkan ID unik di penangan Fungsi Acara

```
namespace Example;
public class SnapstartExample
{
    private Guid _myExecutionEnvironmentGuid;
    public SnapstartExample()
    {
        // This GUID is set for non-restore use cases, such as testing or if
 SnapStart is turned off
        _myExecutionEnvironmentGuid = new Guid();
        // Register the method which will run after each restore. You may need to
 update Amazon.Lambda.Core to see this
        Amazon.Lambda.Core.SnapshotRestore.RegisterAfterRestore(MyAfterRestore);
    }

    private ValueTask MyAfterRestore()
    {
        // After restoring this snapshot to a new execution environment, update the
 GUID
```

```
        _myExecutionEnvironmentGuid = new Guid();
        return ValueTask.CompletedTask;
    }

    public string Handler()
    {
        return $"Hello World! My Execution Environment GUID is
 {_myExecutionEnvironmentGuid}";
    }
}
```

Gunakan generator nomor pseudorandom yang aman secara kriptografis (CSPRNGs)

Ketika SnapStart diaktifkan, Lambda mengambil snapshot dari lingkungan eksekusi fungsi Anda termasuk semua aplikasi dan memori sistem. Ini berarti status internal setiap generator nomor acak (RNG) dipertahankan persis seperti pada waktu snapshot, bersama dengan byte acak apa pun yang aplikasi Anda atau dependensinya telah dihasilkan dan disimpan dalam buffer memori.

Ketika Lambda mengembalikan lingkungan eksekusi dari snapshot, itu menyemai kembali generator nomor acak kernel dari `/dev/random` dan dengan entropi baru. `/dev/urandom` Perangkat lunak yang membaca angka acak langsung dari perangkat ini mempertahankan keacakan dengan SnapStart.

Jika aplikasi Anda bergantung pada keacakan, kami sarankan Anda menggunakan generator nomor acak yang aman secara kriptografis (CSPRNGs). Selain OpenSSL yang disediakan AL, runtime terkelola Lambda yang mendukung SnapStart (Java versi 11+, Python versi 3.12+, dan .NET versi 8+), termasuk CSPRNG bawaan berikut:

- Jawa: `java.security.SecureRandom`
- Python: `random.SystemRandom`
- .NET: `System.Security.Cryptography.RandomNumberGenerator`

> **Note**
>
> Untuk runtime Go Lambda, tidak ada perubahan yang diperlukan jika fungsi Anda menggunakan pustaka standar`crypto/rand`, yang kompatibel dengan snapstart secara default.

Saat Anda mengemas fungsi Anda sebagai gambar kontainer, gambar dasar Anda menentukan kompatibilitas keunikan dengan: SnapStart

- Gambar dasar Lambda untuk runtime terkelola (Java versi 11+, Python versi 3.12+, dan.NET versi 8+) — Kompatibel dengan seperti yang dijelaskan di atas. SnapStart
- Gambar dasar.al2023 yang disediakan — Kompatibel jika runtime bahasa pemrograman Anda memperoleh entropi dari,`/dev/random`, `/dev/urandom` atau build OpenSSL (openssl-snapsafe-libs) SnapStart-compatible yang tercantum dalam daftar paket Amazon Linux 2023. Jika runtime Anda bergantung pada sumber entropi lain, ikuti langkah-langkah yang dijelaskan di bagian "Gambar dasar Anda sendiri" di bawah ini.
- Gambar dasar Anda sendiri — Audit pustaka kriptografi Anda dan gunakan kait runtime pasca-pemulihan untuk membuang status generator nomor acak (RNG) yang di-cache atau byte acak buffer, memastikan perpustakaan membaca entropi baru dari sistem setelah pemulihan. Misalnya, fungsi Rust yang menggunakan `rand::rngs::ThreadRng` harus memanggil `reseed()` setelah restore. Kami menyarankan Anda membuat menggunakan gambar dasar Lambda bila memungkinkan. Khususnya, jika image dasar Anda menyertakan versi OpenSSL sendiri atau fork OpenSSL (seperti BoringSSL atau LibreSSL), beralihlah ke image dasar.al2023 yang disediakan, yang menyertakan build OpenSSL (openssl-snapsafe-libs), atau gunakan libcrypto (-LC). SnapStart-compatible AWSAWS

Versi pustaka minimum yang didukung

AWS pustaka kriptografi secara otomatis mempertahankan keacakan dengan SnapStart memulai dengan versi minimum yang ditentukan dalam tabel berikut. Jika Anda menggunakan pustaka ini dengan Fungsi Peristiwa Lambda, pastikan Anda menggunakan versi minimum berikut atau versi yang lebih baru:

Pustaka	Versi minimum yang didukung (x86)	Versi minimum yang didukung (ARM)
AWS libcrypto (AWS-LC)	1.16.0	1.30.0
AWS libcrypto FIPS	2.0.13	2.0.13

Jika Anda mengemas pustaka kriptografi sebelumnya dengan Fungsi Peristiwa Lambda sebagai dependensi transitif melalui pustaka berikut, pastikan Anda menggunakan versi minimum berikut atau versi yang lebih baru:

Pustaka	Versi minimum yang didukung (x86)	Versi minimum yang didukung (ARM)
AWS SDK for Java 2.x	2.23.20	2.26.12
AWS Runtime Umum untuk Java	0.29.8	0.29.25
Penyedia Crypto Amazon Corretto	2.4.1	2.4.1
Amazon Corretto Penyedia Crypto FIPS	2.4.1	2.4.1

Contoh berikut menunjukkan cara menggunakan CSPrNGs untuk menjamin urutan nomor unik bahkan ketika fungsi dipulihkan dari snapshot.

Java

Example— java.security. SecureRandom

```
import java.security.SecureRandom;
  public class Handler implements RequestHandler<String, String> {
    private static SecureRandom rng = new SecureRandom();
    @Override
    public String handleRequest(String event, Context context) {
      for (int i = 0; i < 10; i++) {
        System.out.println(rng.next());
      }
      return "Hello, World!";
    }
  }
```

Python

Example— acak. SystemRandom

```
import json
import random

secure_rng = random.SystemRandom()

def lambda_handler(event, context):
    random_numbers = [secure_rng.random() for _ in range(10)]

    for number in random_numbers:
        print(number)

    return "Hello, World!"
```

.NET

Example– RandomNumberGenerator

```
using Amazon.Lambda.Core;
using System.Security.Cryptography;
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeriali

namespace DotnetSecureRandom;

public class Function
{
    public string FunctionHandler()
    {
        using (RandomNumberGenerator rng = RandomNumberGenerator.Create())
        {
            byte[] randomUnsignedInteger32Bytes = new byte[4];
            for (int i = 0; i < 10; i++)
            {
                rng.GetBytes(randomUnsignedInteger32Bytes);
                int randomInt32 = BitConverter.ToInt32(randomUnsignedInteger32Bytes,
 0);
                Console.WriteLine("{0:G}", randomInt32);
            }
        }
```

```
        return "Hello World!";
    }
}
```

SnapStart alat pemindaian (hanya Java)

Lambda menyediakan alat pemindaian untuk Java untuk membantu Anda memeriksa kode yang mengasumsikan keunikan. Alat SnapStart pemindaian adalah SpotBugsplugin sumber terbuka yang menjalankan analisis statis terhadap seperangkat aturan. Alat pemindaian membantu mengidentifikasi implementasi kode potensial yang mungkin mematahkan asumsi mengenai keunikan. Untuk petunjuk penginstalan dan daftar pemeriksaan yang dilakukan alat pemindaian, lihat repositori aws-lambda-snapstart-java-rules aktif. GitHub

Untuk mempelajari lebih lanjut tentang menangani keunikan dengan SnapStart, lihat Memulai lebih cepat dengan AWS Lambda SnapStart di AWS Compute Blog.

Menerapkan kode sebelum atau sesudah snapshot Fungsi Peristiwa Lambda

Anda dapat menggunakan kait runtime untuk mengimplementasikan kode sebelum Lambda membuat snapshot atau setelah Lambda melanjutkan fungsi dari snapshot. Runtime hook berguna untuk berbagai tujuan, seperti:

- Pembersihan dan inisialisasi: Sebelum snapshot dibuat, Anda dapat menggunakan hook runtime untuk melakukan pembersihan atau operasi rilis sumber daya. Setelah snapshot dipulihkan, Anda dapat menggunakan hook runtime untuk menginisialisasi ulang sumber daya atau status apa pun yang tidak ditangkap dalam snapshot.
- Konfigurasi dinamis: Anda dapat menggunakan kait runtime untuk memperbarui konfigurasi atau metadata lain secara dinamis sebelum snapshot dibuat atau setelah dipulihkan. Ini dapat berguna jika fungsi Anda perlu beradaptasi dengan perubahan di lingkungan runtime.
- Integrasi eksternal: Anda dapat menggunakan kait runtime untuk berintegrasi dengan layanan atau sistem eksternal, seperti mengirim pemberitahuan atau memperbarui status eksternal, sebagai bagian dari proses pemeriksaan dan pemulihan.
- Penyetelan kinerja: Anda dapat menggunakan kait runtime untuk menyempurnakan urutan startup fungsi Anda, seperti dengan preloading dependensi. Untuk informasi selengkapnya, lihat Penyempurnaan performa.

Halaman-halaman berikut menjelaskan cara menerapkan kait runtime untuk runtime pilihan Anda.

Topik

- Kait SnapStart runtime Lambda untuk Java
- Kait SnapStart runtime Lambda untuk Python
- Kait SnapStart runtime Lambda untuk .NET
- Menerapkan SnapStart kait untuk gambar kontainer

Kait SnapStart runtime Lambda untuk Java

Anda dapat menggunakan kait runtime untuk mengimplementasikan kode sebelum Lambda membuat snapshot atau setelah Lambda melanjutkan Fungsi Peristiwa dari snapshot. Runtime hook tersedia sebagai bagian dari proyek Coordinated Restore at Checkpoint (cRAC) open-source. CRAC sedang dalam pengembangan untuk Open Java Development Kit (OpenJDK). Untuk contoh

cara menggunakan cRAC dengan aplikasi referensi, lihat repositori cRAC aktif. GitHub CraC menggunakan tiga elemen utama:

- `Resource`Antarmuka dengan dua metode, `beforeCheckpoint()` dan`afterRestore()`. Gunakan metode ini untuk mengimplementasikan kode yang ingin Anda jalankan sebelum snapshot dan setelah pemulihan.
- `Context <R extends Resource>`— Untuk menerima pemberitahuan untuk pos pemeriksaan dan pemulihan, a `Resource` harus terdaftar dengan a. `Context`
- `Core`— Layanan koordinasi, yang menyediakan global default `Context` melalui metode statis`Core.getGlobalContext()`.

Untuk informasi selengkapnya tentang `Context` dan`Resource`, lihat Package org.crac di dokumentasi cRAC.

Gunakan langkah-langkah berikut untuk mengimplementasikan kait runtime dengan paket org.crac. Runtime Lambda berisi implementasi konteks cRAC khusus yang memanggil kait runtime Anda sebelum checkpointing dan setelah memulihkan.

Registrasi dan eksekusi runtime hook

Urutan Lambda mengeksekusi kait runtime Anda ditentukan oleh urutan pendaftaran. Pesanan pendaftaran mengikuti urutan impor, definisi, atau eksekusi dalam kode Anda.

- `beforeCheckpoint()`: Dieksekusi dalam urutan terbalik pendaftaran
- `afterRestore()`: Dieksekusi dalam urutan pendaftaran

Pastikan bahwa semua kait terdaftar diimpor dengan benar dan disertakan dalam kode fungsi Anda. Jika Anda mendaftarkan kait runtime dalam file atau modul terpisah, Anda harus memastikan bahwa modul diimpor, baik secara langsung atau sebagai bagian dari paket yang lebih besar, dalam file handler fungsi Anda. Jika file atau modul tidak diimpor dalam penangan fungsi, Lambda mengabaikan kait runtime.

> **Note**
>
> Saat Lambda membuat snapshot, kode inisialisasi Anda dapat berjalan hingga 15 menit. Batas waktu adalah 130 detik atau batas waktu fungsi yang dikonfigurasi (maksimum 900 detik), mana yang lebih tinggi. Kait `beforeCheckpoint()` runtime Anda dihitung terhadap

batas waktu kode inisialisasi. Saat Lambda mengembalikan snapshot, runtime harus dimuat dan kait runtime harus selesai dalam batas `afterRestore()` waktu tunggu (10 detik). Jika tidak, Anda akan mendapatkan SnapStartTimeoutException.

Langkah 1: Perbarui konfigurasi build

Tambahkan `org.crac` dependensi ke konfigurasi build. Contoh berikut menggunakan Gradle. Untuk contoh untuk sistem build lainnya, lihat dokumentasi Apache Maven.

```
dependencies {
    compile group: 'com.amazonaws', name: 'aws-lambda-java-core', version: '1.2.1'
    # All other project dependecies go here:
    # ...
    # Then, add the org.crac dependency:
 implementation group: 'org.crac', name: 'crac', version: '1.4.0'
}
```

Langkah 2: Perbarui penangan Lambda

Lambda Event Function handler adalah metode dalam kode fungsi Anda yang memproses peristiwa. Saat fungsi Anda diaktifkan, Lambda menjalankan metode handler. Fungsi Anda berjalan sampai handler mengembalikan respons, keluar, atau waktu habis.

Untuk informasi selengkapnya, lihat Tentukan Lambda Event Function handler di Java.

Contoh handler berikut menunjukkan cara menjalankan kode sebelum checkpointing (`beforeCheckpoint()`) dan setelah restoring (). `afterRestore()` Handler ini juga mendaftarkan `Resource` ke global yang dikelola runtime. `Context`

Note

Saat Lambda membuat snapshot, kode inisialisasi Anda dapat berjalan hingga 15 menit. Batas waktu adalah 130 detik atau batas waktu fungsi yang dikonfigurasi (maksimum 900 detik), mana yang lebih tinggi. Kait `beforeCheckpoint()` runtime Anda dihitung terhadap batas waktu kode inisialisasi. Saat Lambda mengembalikan snapshot, runtime (JVM) harus dimuat dan kait `afterRestore()` runtime harus selesai dalam batas waktu tunggu (10 detik). Jika tidak, Anda akan mendapatkan SnapStartTimeoutException.

```
...
  import org.crac.Resource;
  import org.crac.Core;
  ...
public class CRaCDemo implements RequestStreamHandler, Resource {
    public CRaCDemo() {
      Core.getGlobalContext().register(this);
    }
    public String handleRequest(String name, Context context) throws IOException {
      System.out.println("Handler execution");
      return "Hello " + name;
    }
    @Override
    public void beforeCheckpoint(org.crac.Context<? extends Resource> context)
        throws Exception {
      System.out.println("Before checkpoint");
    }
    @Override
    public void afterRestore(org.crac.Context<? extends Resource> context)
        throws Exception {
      System.out.println("After restore");
```

`Context`hanya mempertahankan a `WeakReference`ke objek terdaftar. Jika sampah `Resource`dikumpulkan, kait runtime tidak berjalan. Kode Anda harus mempertahankan referensi yang kuat `Resource` untuk menjamin bahwa hook runtime berjalan.

Berikut adalah dua contoh pola yang harus dihindari:

Example— Objek tanpa referensi yang kuat

```
Core.getGlobalContext().register( new MyResource() );
```

Example— Objek kelas anonim

```
Core.getGlobalContext().register( new Resource() {

   @Override
   public void afterRestore(Context<? extends Resource> context) throws Exception {
    // ...
   }

   @Override
```

```
    public void beforeCheckpoint(Context<? extends Resource> context) throws Exception {
     // ...
    }

} );
```

Sebaliknya, pertahankan referensi yang kuat. Dalam contoh berikut, sumber daya terdaftar bukan sampah yang dikumpulkan dan kait runtime berjalan secara konsisten.

Example— Objek dengan referensi yang kuat

```
Resource myResource = new MyResource(); // This reference must be maintained to prevent
 the registered resource from being garbage collected
Core.getGlobalContext().register( myResource );
```

Kait SnapStart runtime Lambda untuk Python

Anda dapat menggunakan kait runtime untuk mengimplementasikan kode sebelum Lambda membuat snapshot atau setelah Lambda melanjutkan fungsi dari snapshot. Kait runtime Python tersedia sebagai bagian dari pustaka Snapshot Restore for Python open-source, yang disertakan dalam runtime terkelola Python. Pustaka ini menyediakan dua dekorator yang dapat Anda gunakan untuk menentukan kait runtime Anda:

- `@register_before_snapshot`: Untuk fungsi yang ingin Anda jalankan sebelum Lambda membuat snapshot.
- `@register_after_restore`: Untuk fungsi yang ingin Anda jalankan saat Lambda melanjutkan fungsi dari snapshot.

Atau, Anda dapat menggunakan metode berikut untuk mendaftarkan callable untuk kait runtime:

- `register_before_snapshot(func, *args, **kwargs)`
- `register_after_restore(func, *args, **kwargs)`

Registrasi dan eksekusi runtime hook

Urutan Lambda mengeksekusi kait runtime Anda ditentukan oleh urutan pendaftaran:

- Sebelum snapshot: Dieksekusi dalam urutan terbalik pendaftaran

- Setelah snapshot: Dieksekusi dalam urutan pendaftaran

Urutan pendaftaran kait runtime tergantung pada bagaimana Anda mendefinisikan kait. Saat menggunakan dekorator (`@register_before_snapshot`dan`@register_after_restore`), urutan pendaftaran mengikuti urutan impor, definisi, atau eksekusi dalam kode Anda. Jika Anda memerlukan kontrol lebih besar atas pesanan pendaftaran, gunakan `register_after_restore()` metode `register_before_snapshot()` dan bukan dekorator.

Pastikan bahwa semua kait terdaftar diimpor dengan benar dan disertakan dalam kode fungsi Anda. Jika Anda mendaftarkan kait runtime dalam file atau modul terpisah, Anda harus memastikan bahwa modul diimpor, baik secara langsung atau sebagai bagian dari paket yang lebih besar, dalam file handler fungsi Anda. Jika file atau modul tidak diimpor dalam penangan fungsi, Lambda mengabaikan kait runtime.

> **Note**
>
> Saat Lambda membuat snapshot, kode inisialisasi Anda dapat berjalan hingga 15 menit. Batas waktu adalah 130 detik atau batas waktu fungsi yang dikonfigurasi (maksimum 900 detik), mana yang lebih tinggi. Kait `@register_before_snapshot` runtime Anda dihitung terhadap batas waktu kode inisialisasi. Saat Lambda mengembalikan snapshot, runtime harus dimuat dan kait runtime harus selesai dalam batas `@register_after_restore` waktu tunggu (10 detik). Jika tidak, Anda akan mendapatkan SnapStartTimeoutException.

Contoh

Contoh handler berikut menunjukkan cara menjalankan kode sebelum checkpointing (`@register_before_snapshot`) dan setelah restoring (). `@register_after_restore`

```
from snapshot_restore_py import register_before_snapshot, register_after_restore

def lambda_handler(event, context):
    # Handler code

@register_before_snapshot
def before_checkpoint():
    # Logic to be executed before taking snapshots

@register_after_restore
```

```
def after_restore():
    # Logic to be executed after restore
```

Untuk contoh lainnya, lihat Pemulihan Snapshot untuk Python di AWS GitHub repositori.

Kait SnapStart runtime Lambda untuk .NET

Anda dapat menggunakan kait runtime untuk mengimplementasikan kode sebelum Lambda membuat snapshot atau setelah Lambda melanjutkan Fungsi Peristiwa dari snapshot. .NET runtime hooks tersedia sebagai bagian dari Amazon.Lambda.Corepaket (versi 2.5.0 atau yang lebih baru). Pustaka ini menyediakan dua metode yang dapat Anda gunakan untuk menentukan kait runtime Anda:

- `RegisterBeforeSnapshot()`: Kode untuk dijalankan sebelum pembuatan snapshot
- `RegisterAfterSnapshot()`: Kode untuk dijalankan setelah melanjutkan fungsi dari snapshot

> **Note**
>
> Jika Anda menggunakan kerangka kerja Anotasi Lambda untuk.NET, tingkatkan ke Amazon.Lambda.Annotationsversi 1.6.0 atau yang lebih baru untuk memastikan kompatibilitas dengan. SnapStart

Registrasi dan eksekusi runtime hook

Daftarkan kait Anda dalam kode inisialisasi Anda. Pertimbangkan panduan berikut berdasarkan model eksekusi fungsi Lambda Anda:

- Untuk pendekatan perakitan yang dapat dieksekusi, daftarkan kait Anda sebelum Anda memulai bootstrap Lambda dengan. `RunAsync`
- Untuk pendekatan perpustakaan kelas, daftarkan kait Anda di konstruktor kelas handler.
- Untuk aplikasi ASP.NET Core, daftarkan kait Anda sebelum memanggil `WebApplications.Run` metode.

Untuk mendaftarkan kait runtime SnapStart di .NET, gunakan metode berikut:

```
Amazon.Lambda.Core.SnapshotRestore.RegisterBeforeSnapshot(BeforeCheckpoint);
```

```
Amazon.Lambda.Core.SnapshotRestore.RegisterAfterRestore(AfterCheckpoint);
```

Ketika beberapa jenis hook terdaftar, urutan Lambda mengeksekusi kait runtime Anda ditentukan oleh urutan pendaftaran:

- `RegisterBeforeSnapshot()`: Dieksekusi dalam urutan terbalik pendaftaran
- `RegisterAfterSnapshot()`: Dieksekusi dalam urutan pendaftaran

> **Note**
>
> Saat Lambda membuat snapshot, kode inisialisasi Anda dapat berjalan hingga 15 menit. Batas waktu adalah 130 detik atau batas waktu fungsi yang dikonfigurasi (maksimum 900 detik), mana yang lebih tinggi. Kait `RegisterBeforeSnapshot()` runtime Anda dihitung terhadap batas waktu kode inisialisasi. Saat Lambda mengembalikan snapshot, runtime harus dimuat dan kait runtime harus selesai dalam batas `RegisterAfterSnapshot()` waktu tunggu (10 detik). Jika tidak, Anda akan mendapatkan SnapStartTimeoutException.

Contoh

Contoh fungsi berikut menunjukkan bagaimana menjalankan kode sebelum checkpointing (`RegisterBeforeSnapshot`) dan setelah restoring (). `RegisterAfterRestore`

```
public class SampleClass
{
    public SampleClass()
    {
        Amazon.Lambda.Core.SnapshotRestore.RegisterBeforeSnapshot(BeforeCheckpoint);
        Amazon.Lambda.Core.SnapshotRestore.RegisterAfterRestore(AfterCheckpoint);
    }

    private ValueTask BeforeCheckpoint()
    {
        // Add logic to be executed before taking the snapshot
        return ValueTask.CompletedTask;
    }

    private ValueTask AfterCheckpoint()
    {
        // Add logic to be executed after restoring the snapshot
```

```
        return ValueTask.CompletedTask;
    }

    public APIGatewayProxyResponse FunctionHandler(APIGatewayProxyRequest request,
 ILambdaContext context)
    {
        // Add business logic

        return new APIGatewayProxyResponse
        {
            StatusCode = 200
        };
    }
}
```

Menerapkan SnapStart kait untuk gambar kontainer

Gambaran umum

Saat menggunakan SnapStart dengan fungsi image container, runtime perlu mengoordinasikan siklus hidup fungsi dan memanggil kait before-snapshot dan after-restore selama fase siklus hidup yang sesuai. Kait ini memungkinkan Anda menjalankan logika kustom (seperti menyegarkan kredensyal atau menyemai ulang generator nomor acak) pada titik-titik dalam siklus hidup snapshot-and-restore. Jika Anda menggunakan runtime terkelola yang didukung atau gambar dasar yang sesuai untuk runtime tersebut (Java versi 11+, Python versi 3.12+, dan.NET versi 8+) Lambda mengoordinasikan siklus hidup untuk Anda. SnapStart Daftarkan hook Anda melalui API yang dijelaskan diMenerapkan kode sebelum atau sesudah snapshot Fungsi Peristiwa Lambda.

Jika Anda menggunakan gambar wadah dasar Anda sendiri, Klien Antarmuka Runtime (RIC), atau gambar dasar Lambda untuk disediakan.al2023,, atau Ruby Node.js, ikuti langkah-langkah di halaman ini untuk digunakan. SnapStart

Prasyarat

Saat Lambda mengembalikan fungsi dari snapshot, status apa pun yang ditentukan selama inisialisasi, seperti generator nomor acak, ID unik, dan kredensyal cache, akan dibagikan di semua lingkungan eksekusi yang dipulihkan dari snapshot tersebut. Sebelum menggunakan SnapStart dengan Fungsi Peristiwa Lambda berbasis gambar kontainer Anda, tinjau Menangani keunikan dengan Lambda SnapStart dan pastikan bahwa persyaratan yang diuraikan dalam Gunakan generator nomor pseudorandom yang aman secara kriptografis (CSPrNGs) terpenuhi.

Setelah memvalidasi bahwa persyaratan terpenuhi, pilih salah satu dari dua opsi berikut:

1. Opsi 1: Jika Anda memerlukan kait sebelum-snapshot dan sesudah pemulihan untuk menjalankan logika khusus saat snapshot fungsi Anda dilanjutkan, ikuti petunjuk di bagian. Menerapkan SnapStart kait siklus hidup
2. Opsi 2: Jika Anda tidak memerlukan kait ini, aktifkan SnapStart gambar kontainer ini dengan menentukan label berikut dalam Dockerfile Anda:

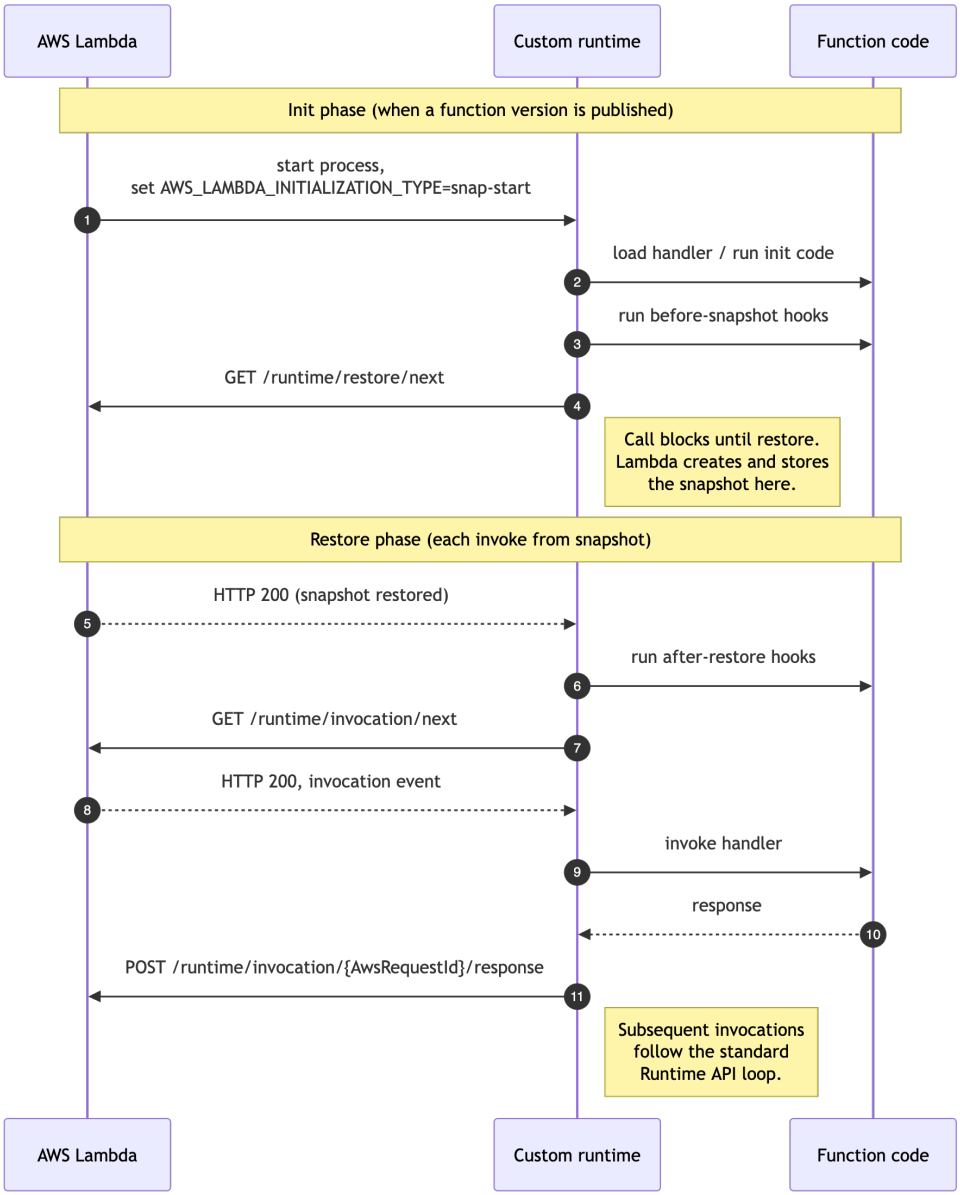
```
LABEL com.amazonaws.lambda.feature.snapstart="Allow"
```

Jika image container Anda tidak mengimplementasikan `/restore/next` API atau menyertakan label, versi publish akan gagal.

> **Note**
>
> Prasyarat ini tidak diperlukan jika Anda menggunakan gambar dasar terkelola Lambda untuk Java (versi 11+), Python (versi 3.12+), dan .NET (versi 8+) karena sudah mengoordinasikan siklus hidup dan memberikan persyaratan keunikan. SnapStart

Ikhtisar siklus hidup

Diagram berikut menunjukkan urutan panggilan Runtime API yang diharapkan dibuat oleh runtime SnapStart kustom. Semua panggilan adalah bagian dari kontrak Runtime API; sedangkan panggilan #3, #4, #5, dan #6 khusus untuk SnapStart.



Menerapkan SnapStart kait siklus hidup

Untuk menggunakan SnapStart fungsi image container Anda, ikuti langkah-langkah di bawah ini:

1. Jalankan kait sebelum snapshot dan picu proses snapshotting: Sebagai langkah terakhir dari kode inisialisasi fungsi Anda, jalankan kait sebelum snapshot jika diperlukan dan picu proses snapshotting. Lakukan langkah-langkah ini hanya jika SnapStart diaktifkan, dengan memeriksa bahwa nilai variabel `AWS_LAMBDA_INITIALIZATION_TYPE` lingkungan diatur ke`snap-start`. Jalankan kait sebelum snapshot terdaftar Anda, lalu panggil `GET /runtime/restore/next` untuk memicu proses snapshotting. Jika hook before-snapshot melempar atau mengembalikan

kesalahan, runtime akan memposting kesalahan ke titik akhir. `/runtime/init/error` Lihat contoh pseudo-code di bawah ini:

```
# After all initialization code has finished:

   READ initialization_type FROM environment variable
 "AWS_LAMBDA_INITIALIZATION_TYPE"

   IF initialization_type IS "snap-start" THEN

       TRY
           EXECUTE registered before-snapshot hooks
       ON ERROR
           POST error to /runtime/init/error
               SET header  Lambda-Runtime-Function-Error-Type  TO
 <Category.Reason>
               SET body    TO  { errorMessage, errorType, stackTrace }
           EXIT process with non-zero code

       # Signal readiness for snapshot
       SEND GET request to /runtime/restore/next
       # The request blocks until Lambda restores the execution environment from
 the snapshot, then returns HTTP 200.

   END IF
```

> **Note**
>
> Fase init dan kait sebelum snapshot berbagi batas waktu gabungan. `max(function_timeout, 130 seconds)` Jika batas ini terlampaui, Lambda gagal PublishVersion permintaan. Selain itu, seperti`/runtime/invocation/next`, `/runtime/restore/next` panggilan adalah panggilan pemblokiran. Ini memblokir sampai Lambda mengembalikan lingkungan eksekusi dari snapshot.

2. Jalankan after-restore hooks, lalu masukkan loop pemanggilan. Saat `GET /runtime/restore/next` mengembalikan 200, runtime Anda harus mengeksekusi hook after-restore yang terdaftar sebelum melanjutkan ke loop pemanggilan. Jika hook after-restore gagal, laporkan kesalahannya ke`/runtime/restore/error`. Setelah kait after-restore selesai, masukkan loop pemanggilan standar dengan menelepon. `GET /runtime/invocation/next` Mulai saat ini,

perilaku identik dengan fungsi yang tidak digunakan SnapStart. Lihat contoh pseudo-code di bawah ini:

```
# After the snapshot has been restored
# (i.e., GET /runtime/restore/next has returned HTTP 200):

    TRY
        EXECUTE registered after-restore hooks
    ON ERROR
        POST error to /runtime/restore/error
            SET header  Lambda-Runtime-Function-Error-Type  TO  <Category.Reason>
            SET body    TO  { errorMessage, errorType, stackTrace }

# Proceed to the invoke loop
       GET /runtime/invocation/next
```

Penanganan kesalahan

Jika hook gagal, runtime harus melaporkan kesalahan ke titik akhir API yang sesuai dan keluar dari proses. Tabel di bawah ini merangkum perilaku untuk setiap fase:

Fase	Titik akhir API kesalahan	Apa yang terjadi pada kegagalan
Init/sebelum-snapshot	`POST /runtime/init/error`	Lambda gagal permintaan. PublishVersion Keluar dari proses.
After-restore	`POST /runtime/restore/error`	Lambda gagal dalam pemanggilan dalam penerbangan dan merobohkan lingkungan eksekusi. Keluar dari proses.

Untuk kedua titik akhir API, setel `Lambda-Runtime-Function-Error-Type` header ke nilai dalam format `<Category.Reason>` (misalnya, `Runtime.BeforeSnapshotError` atau`Runtime.AfterRestoreError`). Sertakan badan kesalahan dengan`errorMessage`,`errorType`, dan opsional`stackTrace`.

Untuk spesifikasi endpoint API lengkap dan kode respons, lihat Kesalahan inisialisasi dan Mengembalikan kesalahan (hanya berlaku untuk SnapStart) di referensi Runtime API.

Pemantauan untuk Lambda SnapStart

Anda dapat memantau SnapStart fungsi Lambda Anda menggunakan Amazon CloudWatch, AWS X-Ray, dan. Mengakses data telemetri real-time untuk ekstensi menggunakan API Telemetri

> **Note**
>
> Variabel `AWS_LAMBDA_LOG_GROUP_NAME` dan `AWS_LAMBDA_LOG_STREAM_NAME` lingkungan tidak tersedia dalam fungsi Lambda SnapStart .

Memahami perilaku pencatatan dan penagihan dengan SnapStart

Ada beberapa perbedaan dengan format aliran CloudWatch log untuk SnapStart fungsi:

- Log inisialisasi — Ketika lingkungan eksekusi baru dibuat, `REPORT` tidak menyertakan `Init Duration` bidang. Itu karena Lambda menginisialisasi SnapStart fungsi saat Anda membuat versi, bukan selama pemanggilan fungsi. Untuk SnapStart fungsi, `Init Duration` bidang ada dalam `INIT_REPORT` catatan. Catatan ini menunjukkan detail durasi untukFase inisialisasi, termasuk durasi kait `beforeCheckpoint` runtime apa pun.
- Log pemanggilan — Ketika lingkungan eksekusi baru dibuat, `REPORT` termasuk `Restore Duration` dan `Billed Restore Duration` bidang:
  - `Restore Duration`: Waktu yang dibutuhkan Lambda untuk memulihkan snapshot, memuat runtime, dan menjalankan kait runtime setelah pemulihan. Proses memulihkan snapshot dapat mencakup waktu yang dihabiskan untuk aktivitas di luar microVM. Kali ini dilaporkan dalam`Restore Duration`.
  - `Billed Restore Duration`: Waktu yang dibutuhkan Lambda untuk memuat runtime dan menjalankan kait runtime setelah pemulihan.

> **Note**
>
> Seperti semua Fungsi Peristiwa Lambda, biaya durasi berlaku untuk kode yang berjalan di penangan fungsi. Untuk SnapStart fungsi, biaya durasi juga berlaku untuk kode inisialisasi yang dideklarasikan di luar handler, waktu yang diperlukan untuk memuat runtime, dan kode apa pun yang berjalan di hook runtime.

Durasi awal dingin adalah jumlah dari `Restore Duration` +`Duration`.

Contoh berikut adalah kueri Lambda Insights yang mengembalikan persentil latensi untuk fungsi. SnapStart Untuk informasi selengkapnya tentang kueri Lambda Insights, lihat. Contoh alur kerja menggunakan kueri untuk memecahkan masalah Fungsi Peristiwa

```
filter @type = "REPORT"
  | parse @log /\d+:\/aws\/lambda\/(?<function>.*)/
  | parse @message /Restore Duration: (?<restoreDuration>.*?) ms/
  | stats
count(*) as invocations,
pct(@duration+coalesce(@initDuration,0)+coalesce(restoreDuration,0), 50) as p50,
pct(@duration+coalesce(@initDuration,0)+coalesce(restoreDuration,0), 90) as p90,
pct(@duration+coalesce(@initDuration,0)+coalesce(restoreDuration,0), 99) as p99,
pct(@duration+coalesce(@initDuration,0)+coalesce(restoreDuration,0), 99.9) as p99.9
group by function, (ispresent(@initDuration) or ispresent(restoreDuration)) as
 coldstart
  | sort by coldstart desc
```

X-Ray penelusuran aktif untuk SnapStart

Anda dapat menggunakan X-Rayuntuk melacak permintaan ke fungsi Lambda SnapStart . Ada beberapa perbedaan dengan X-Ray subsegmen untuk SnapStart fungsi:

- Tidak ada `Initialization` subsegmen untuk SnapStart fungsi.
- `Restore`Subsegmen menunjukkan waktu yang dibutuhkan Lambda untuk memulihkan snapshot, memuat runtime, dan menjalankan kait runtime setelah pemulihan. Proses memulihkan snapshot dapat mencakup waktu yang dihabiskan untuk aktivitas di luar microVM. Kali ini dilaporkan di `Restore` subsegmen. Anda tidak dikenakan biaya untuk waktu yang dihabiskan di luar microVM untuk memulihkan snapshot.

Acara API telemetri untuk SnapStart

Lambda mengirimkan SnapStart peristiwa berikut ke: API Telemetri

- `platform.RestoreStart`— Menunjukkan waktu ketika `Restore`fase dimulai.
- `platform.restore RuntimeDone`— Menunjukkan apakah `Restore` fase itu berhasil. Lambda mengirimkan pesan ini saat runtime mengirimkan permintaan API `restore/next` runtime. Ada tiga kemungkinan status: sukses, gagal, dan batas waktu.

- `Platform.RestoreReport`— Menunjukkan berapa lama `Restore` fase berlangsung dan berapa milidetik Anda ditagih selama fase ini.

Amazon API Gateway dan metrik URL fungsi

Jika Anda membuat API web menggunakan API Gateway, Anda dapat menggunakan IntegrationLatencymetrik untuk mengukur latensi ujung ke ujung (waktu antara saat API Gateway menyampaikan permintaan ke backend dan saat menerima respons dari backend).

Jika Anda menggunakan URL fungsi Lambda, maka Anda dapat menggunakan UrlRequestLatencymetrik untuk mengukur latensi ujung ke ujung (waktu antara saat URL fungsi menerima permintaan dan saat URL fungsi mengembalikan respons).

Model keamanan untuk Lambda SnapStart

AWS Lambda SnapStart mendukung enkripsi saat istirahat. Lambda mengenkripsi snapshot dengan file. AWS KMS key Secara default, Lambda menggunakan file. Kunci yang dikelola AWS Jika perilaku default ini sesuai dengan alur kerja Anda, maka Anda tidak perlu menyiapkan hal lain.

Jika tidak, Anda dapat menggunakan `--kms-key-arn` opsi dalam perintah create-function atau update-function-configuration untuk menyediakan kunci terkelola pelanggan. AWS KMS Anda dapat melakukan ini untuk mengontrol rotasi kunci KMS atau untuk memenuhi persyaratan organisasi Anda untuk mengelola kunci KMS. Kunci yang dikelola pelanggan dikenakan AWS KMS biaya standar. Untuk informasi selengkapnya, lihat harga AWS Key Management Service.

Jika Fungsi Acara Anda juga merupakan fungsi yang tahan lama, kunci terkelola pelanggan yang mengenkripsi SnapStart snapshot tidak tergantung pada kunci yang dikelola pelanggan yang mengenkripsi data eksekusi yang tahan lama. Untuk informasi selengkapnya, lihat Mengenkripsi AWS Lambda data eksekusi tahan lama.

Saat Anda menghapus versi SnapStart fungsi atau fungsi, semua `Invoke` permintaan ke fungsi atau versi fungsi tersebut gagal. Lambda menghapus semua sumber daya yang terkait dengan snapshot yang dihapus sesuai dengan Peraturan Perlindungan Data Umum (GDPR).

Maksimalkan kinerja Lambda SnapStart

Topik

- Penyempurnaan performa
- Praktik terbaik jaringan

Penyempurnaan performa

Untuk memaksimalkan manfaat SnapStart, pertimbangkan rekomendasi pengoptimalan kode berikut untuk runtime Anda.

> **Note**
>
> SnapStart bekerja paling baik saat digunakan dengan pemanggilan fungsi dalam skala besar. Fungsi yang jarang dipanggil mungkin tidak mengalami peningkatan kinerja yang sama.

Java

Untuk memaksimalkan manfaat SnapStart, kami menyarankan Anda memuat dependensi terlebih dahulu dan menginisialisasi sumber daya yang berkontribusi pada latensi startup dalam kode inisialisasi Anda, bukan di penangan Fungsi Peristiwa. Ini memindahkan latensi yang terkait dengan pemuatan kelas berat keluar dari jalur pemanggilan, mengoptimalkan kinerja startup dengan. SnapStart

Jika Anda tidak dapat memuat dependensi atau sumber daya selama inisialisasi, sebaiknya Anda memuatnya terlebih dahulu dengan pemanggilan dummy. Untuk melakukan ini, perbarui kode fungsi handler, seperti yang ditunjukkan pada contoh berikut dari fungsi pet store pada GitHub repositori AWS Labs.

```
private static SpringLambdaContainerHandler<AwsProxyRequest, AwsProxyResponse> handler;
  static {
      try {
          handler =
 SpringLambdaContainerHandler.getAwsProxyHandler(PetStoreSpringAppConfig.class);

          // Use the onStartup method of the handler to register the custom filter
          handler.onStartup(servletContext -> {
```

```
            FilterRegistration.Dynamic registration =
servletContext.addFilter("CognitoIdentityFilter", CognitoIdentityFilter.class);
            registration.addMappingForUrlPatterns(EnumSet.of(DispatcherType.REQUEST),
false, "/*");
        });

        // Send a fake Amazon API Gateway request to the handler to load classes
ahead of time
        ApiGatewayRequestIdentity identity = new ApiGatewayRequestIdentity();
        identity.setApiKey("foo");
        identity.setAccountId("foo");
        identity.setAccessKey("foo");

        AwsProxyRequestContext reqCtx = new AwsProxyRequestContext();
        reqCtx.setPath("/pets");
        reqCtx.setStage("default");
        reqCtx.setAuthorizer(null);
        reqCtx.setIdentity(identity);

        AwsProxyRequest req = new AwsProxyRequest();
        req.setHttpMethod("GET");
        req.setPath("/pets");
        req.setBody("");
        req.setRequestContext(reqCtx);

        Context ctx = new TestContext();
        handler.proxy(req, ctx);


    } catch (ContainerInitializationException e) {
        // if we fail here. We re-throw the exception to force another cold start
        e.printStackTrace();
        throw new RuntimeException("Could not initialize Spring framework", e);
    }
 }
```

Python

Untuk memaksimalkan manfaat SnapStart, fokuslah pada organisasi kode yang efisien dan manajemen sumber daya dalam fungsi Python Anda. Sebagai pedoman umum, lakukan tugas komputasi berat selama fase inisialisasi. Pendekatan ini memindahkan operasi yang memakan waktu keluar dari jalur pemanggilan, meningkatkan kinerja fungsi secara keseluruhan. Untuk menerapkan strategi ini secara efektif, kami merekomendasikan praktik terbaik berikut:

- Impor dependensi di luar fungsi handler.
- Buat `boto3` instance di luar handler.
- Inisialisasi sumber daya statis atau konfigurasi sebelum handler dipanggil.
- Pertimbangkan untuk menggunakan hook runtime sebelum snapshot untuk tugas intensif sumber daya seperti mengunduh file eksternal, kerangka kerja pra-pemuatan seperti Django, atau memuat model pembelajaran mesin.

Example— Optimalkan Fungsi Acara Python untuk SnapStart

```
# Import all dependencies outside of Lambda handler
from snapshot_restore_py import register_before_snapshot
import boto3
import pandas
import pydantic

# Create S3 and SSM clients outside of Lambda handler
s3_client = boto3.client("s3")

# Register the function to be called before snapshot
@register_before_snapshot
def download_llm_models():
    # Download an object from S3 and save to tmp
    # This files will persist in this snapshot
    with open('/tmp/FILE_NAME', 'wb') as f:
        s3_client.download_fileobj('amzn-s3-demo-bucket', 'OBJECT_NAME', f)
    ...

def lambda_handler(event, context):
    ...
```

.NET

Untuk mengurangi kompilasi just-in-time (JIT) dan waktu pemuatan perakitan, pertimbangkan untuk memanggil penangan Fungsi Acara Anda dari kait runtime. `RegisterBeforeCheckpoint` Karena cara kerja kompilasi berjenjang .NET, Anda akan mendapatkan hasil yang optimal dengan memanggil handler beberapa kali, seperti yang ditunjukkan pada contoh berikut.

> **Important**
>
> Pastikan bahwa pemanggilan fungsi dummy Anda tidak menghasilkan efek samping yang tidak diinginkan, seperti memulai transaksi bisnis.

Example

```
public class Function
{
    public Function()
    {
        Amazon.Lambda.Core.SnapshotRestore.RegisterBeforeSnapshot(FunctionWarmup);
    }

    // Warmup method that calls the function handler before snapshot to warm up
 the .NET code and runtime.
    // This speeds up future cold starts after restoring from a snapshot.

    private async ValueTask FunctionWarmup()
    {
        var request = new APIGatewayProxyRequest
        {
            Path = "/heathcheck",
            HttpMethod = "GET"
        };

        for (var i = 0; i < 10; i++)
        {
            await FunctionHandler(request, null);
        }
    }

    public async Task<APIGatewayProxyResponse> FunctionHandler(APIGatewayProxyRequest
 request, ILambdaContext context)
    {
        //
        // Process HTTP request
        //

        var response = new APIGatewayProxyResponse
        {
            StatusCode = 200
```

```
        };

        return await Task.FromResult(response);
    }
}
```

Praktik terbaik jaringan

Status koneksi yang dibuat oleh Event Function Anda selama fase inisialisasi tidak dijamin saat Lambda melanjutkan fungsi Anda dari snapshot. Dalam kebanyakan kasus, koneksi jaringan yang dibuat AWS SDK secara otomatis dilanjutkan. Untuk koneksi lain, kami merekomendasikan praktik terbaik berikut.

Re-establish koneksi jaringan

Selalu buat kembali koneksi jaringan Anda saat fungsi Anda dilanjutkan dari snapshot. Kami menyarankan Anda membangun kembali koneksi jaringan di function handler. Atau, Anda dapat menggunakan hook runtime setelah pemulihan.

Jangan gunakan nama host sebagai pengenal lingkungan eksekusi yang unik

Kami merekomendasikan untuk tidak menggunakan `hostname` untuk mengidentifikasi lingkungan eksekusi Anda sebagai node atau wadah unik dalam aplikasi Anda. Dengan SnapStart, satu snapshot digunakan sebagai status awal untuk beberapa lingkungan eksekusi. Semua lingkungan eksekusi mengembalikan `hostname` nilai yang sama untuk `InetAddress.getLocalHost()` (Java), `socket.gethostname()` (Python), dan `Dns.GetHostName()` (.NET). Untuk aplikasi yang memerlukan identitas atau `hostname` nilai lingkungan eksekusi yang unik, sebaiknya Anda membuat ID unik di pengendali fungsi. Atau, gunakan hook runtime setelah pemulihan untuk menghasilkan ID unik, lalu gunakan ID unik sebagai pengenal untuk lingkungan eksekusi.

Hindari mengikat koneksi ke port sumber tetap

Kami menyarankan Anda menghindari pengikatan koneksi jaringan ke port sumber tetap. Koneksi dibuat kembali ketika fungsi dilanjutkan dari snapshot, dan koneksi jaringan yang terikat ke port sumber tetap mungkin gagal.

Hindari menggunakan cache DNS Java

Fungsi Peristiwa Lambda sudah menyimpan respons DNS cache. Jika Anda menggunakan cache DNS lain SnapStart, Anda mungkin mengalami batas waktu koneksi saat fungsi dilanjutkan dari snapshot.

`java.util.logging.Logger`Kelas secara tidak langsung dapat mengaktifkan cache DNS JVM. Untuk mengganti pengaturan default, setel networkaddress.cache.ttl ke 0 sebelum menginisialisasi. `logger` Contoh:

```
public class MyHandler {
  // first set TTL property
  static{
   java.security.Security.setProperty("networkaddress.cache.ttl" , "0");
  }
 // then instantiate logger
  var logger = org.apache.logging.log4j.LogManager.getLogger(MyHandler.class);
}
```

Untuk mencegah `UnknownHostException` kegagalan dalam runtime Java 11, kami sarankan pengaturan `networkaddress.cache.negative.ttl` ke 0. Di Java 17 dan runtime yang lebih baru, langkah ini tidak diperlukan. Anda dapat mengatur properti ini untuk Fungsi Peristiwa Lambda dengan variabel `AWS_LAMBDA_JAVA_NETWORKADDRESS_CACHE_NEGATIVE_TTL=0` lingkungan.

Menonaktifkan cache DNS JVM tidak menonaktifkan cache DNS terkelola Lambda.

Memecahkan masalah SnapStart kesalahan untuk Fungsi Peristiwa Lambda

Halaman ini membahas masalah umum yang terjadi saat menggunakan Lambda SnapStart, termasuk kesalahan pembuatan snapshot, kesalahan batas waktu, dan kesalahan layanan internal.

SnapStartNotReadyException

Kesalahan: Terjadi kesalahan (SnapStartNotReadyException) saat memanggil operasi Invoke20150331: Lambda menginisialisasi Fungsi Acara Anda. Ini akan siap untuk dipanggil setelah status fungsi Anda menjadi AKTIF.

Penyebab umum

Kesalahan ini terjadi ketika Anda mencoba memanggil versi fungsi yang ada dalam `Inactive` status. Versi fungsi Anda menjadi `Inactive` saat belum dipanggil selama 14 hari atau saat Lambda mendaur ulang lingkungan eksekusi secara berkala

Resolusi

Tunggu hingga versi fungsi mencapai `Active` status, lalu panggil lagi.

SnapStartTimeoutException

Masalah: Anda menerima `SnapStartTimeoutException` ketika Anda mencoba untuk memanggil versi SnapStart fungsi.

Penyebab umum

Selama fase Restore, Lambda mengembalikan runtime Java dan menjalankan hook runtime setelah pemulihan. Jika hook runtime after-restore berjalan lebih dari 10 detik, `Restore` fase akan habis dan Anda mendapatkan kesalahan saat mencoba menjalankan fungsi tersebut. Masalah koneksi jaringan dan kredensional juga dapat menyebabkan batas waktu `Restore` fase.

Resolusi

Periksa CloudWatch log fungsi untuk kesalahan batas waktu yang terjadi selama fase Restore. Pastikan semua kait setelah pemulihan selesai dalam waktu kurang dari 10 detik.

Example CloudWatch log

```
{ "cause": "Lambda couldn't restore the snapshot within the timeout limit. (Service:
 Lambda, Status Code: 408, Request ID: 11a222c3-410f-427c-ab22-931d6bcbf4f2)", "error":
 "Lambda.SnapStartTimeoutException"}
```

500 Kesalahan Layanan Internal

Kesalahan: Lambda tidak dapat membuat snapshot baru karena Anda telah mencapai batas pembuatan snapshot bersamaan.

Penyebab umum

Kesalahan 500 adalah kesalahan internal dalam layanan Lambda itu sendiri, bukan masalah dengan fungsi atau kode Anda. Kesalahan ini sering terputus-putus.

Resolusi

Coba publikasikan versi fungsi lagi.

401 Tidak Sah

Kesalahan: Token sesi buruk atau kunci header

Penyebab umum

Kesalahan ini terjadi ketika Anda menggunakan AWS Systems Manager Parameter Store dan AWS Secrets Manager ekstensi dengan Lambda SnapStart. Saat fungsi Anda digunakan SnapStart, ia memperoleh kredensil dari titik akhir kontainer, dan Lambda tidak menyetel variabel lingkungan. `AWS_SESSION_TOKEN` Jika fungsi Anda menyetel `X-Aws-Parameters-Secrets-Token` header dari variabel lingkungan tersebut, header kosong dan ekstensi menolak permintaan.

Resolusi

Untuk memperbaiki kesalahan ini, gunakan rantai penyedia kredensi AWS SDK untuk mengambil token sesi alih-alih membaca variabel `AWS_SESSION_TOKEN` lingkungan secara langsung. Atur `X-Aws-Parameters-Secrets-Token` header ke token yang dikembalikan oleh rantai penyedia. Rantai penyedia kredensi mengembalikan token sesi yang benar di semua mode inisialisasi, termasuk. SnapStart Untuk kode sampel, lihat Gunakan rahasia Secrets Manager di Lambda Event Functions.

UnknownHostException (Jawa)

Kesalahan: Tidak dapat menjalankan permintaan HTTP: Sertifikat untuk `abc.us-east-1.amazonaws.com` tidak cocok dengan nama alternatif subjek mana pun.

Penyebab umum

Fungsi Peristiwa Lambda sudah menyimpan respons DNS cache. Jika Anda menggunakan cache DNS lain SnapStart, Anda mungkin mengalami batas waktu koneksi saat fungsi dilanjutkan dari snapshot.

Resolusi

Untuk mencegah `UnknownHostException` kegagalan dalam runtime Java 11, kami sarankan pengaturan `networkaddress.cache.negative.ttl` ke 0. Di Java 17 dan runtime yang lebih baru, langkah ini tidak diperlukan. Anda dapat mengatur properti ini untuk Fungsi Peristiwa Lambda dengan variabel `AWS_LAMBDA_JAVA_NETWORKADDRESS_CACHE_NEGATIVE_TTL=0` lingkungan.

Kegagalan pembuatan snapshot

Kesalahan: tidak AWS Lambda dapat menjalankan SnapStart fungsi Anda. Jika kesalahan ini berlanjut, periksa CloudWatch log fungsi Anda untuk kesalahan inisialisasi.

Resolusi

Tinjau CloudWatch log Amazon fungsi Anda untuk mengetahui batas waktu kait runtime sebelum pos pemeriksaan. Anda juga dapat mencoba menerbitkan versi fungsi baru, yang terkadang dapat menyelesaikan masalah.

Latensi pembuatan snapshot

Masalah: Saat Anda mempublikasikan versi fungsi baru, fungsi tetap dalam `Pending` status untuk waktu yang lama.

Penyebab umum

Saat Lambda membuat snapshot, kode inisialisasi Anda dapat berjalan hingga 15 menit. Batas waktu adalah 130 detik atau batas waktu fungsi yang dikonfigurasi (maksimum 900 detik), mana yang lebih tinggi.

Jika fungsi Anda terpasang ke VPC, Lambda mungkin juga perlu membuat antarmuka jaringan sebelum fungsinya menjadi. `Active` Jika Anda mencoba memanggil versi fungsi saat

fungsinya`Pending`, Anda mungkin mendapatkan 409`ResourceConflictException`. Jika fungsi dipanggil menggunakan titik akhir Amazon API Gateway, Anda mungkin mendapatkan kesalahan 500 di API Gateway.

Resolusi

Tunggu setidaknya 15 menit hingga versi fungsi diinisialisasi sebelum memanggilnya.

Isolasi penyewa

Gunakan mode isolasi penyewa saat Anda memerlukan pemrosesan permintaan terisolasi untuk pengguna akhir individu atau penyewa yang menjalankan Fungsi Peristiwa Lambda, yang sebelumnya dikenal sebagai fungsi Lambda. Kemampuan ini menyederhanakan pembuatan aplikasi multi-tenant yang memproses kode atau data khusus penyewa, seperti platform SaaS untuk otomatisasi alur kerja atau eksekusi kode, dengan menghilangkan kebutuhan untuk mengelola sumber daya fungsi khusus penyewa dan meminta logika perutean.

Multi-tenant aplikasi memiliki persyaratan isolasi yang ketat saat menjalankan kode atau memproses data untuk penyewa individu atau pengguna akhir. Dengan mode isolasi penyewa, Lambda menggunakan pengenal penyewa yang ditentukan pelanggan untuk merutekan permintaan ke lingkungan eksekusi fungsi yang mendasarinya, memastikan bahwa lingkungan eksekusi fungsi hanya digunakan untuk melayani pemanggilan dari pengguna akhir atau penyewa yang ditentukan. Lingkungan eksekusi fungsi Lambda menggunakan virtualisasi Firecracker untuk menyediakan isolasi beban kerja.

Ketika Fungsi Peristiwa yang menggunakan mode isolasi penyewa menerima panggilan dengan pengenal penyewa, Lambda pertama-tama mencoba menemukan lingkungan eksekusi yang tersedia yang terkait dengan pengenal penyewa tersebut. Jika tidak ada lingkungan eksekusi, Lambda membuat dan menetapkan lingkungan eksekusi baru untuk penyewa tersebut. Saat pemanggilan fungsi dengan pengenal penyewa yang ditentukan meningkat, Lambda menemukan atau membuat lingkungan eksekusi baru seperlunya.

Untuk mempelajari cara menggunakan isolasi penyewa, lihatthe section called “Mengaktifkan isolasi penyewa”. Untuk informasi tentang mengakses konteks penyewa dalam kode Anda, lihat. the section called “Mengakses dalam kode” Untuk memantau fungsi yang terisolasi penyewa, lihat. the section called “Memantau” Untuk bantuan menyelesaikan masalah, lihatthe section called “Pemecahan masalah”.

Topik

- Kapan menggunakan mode isolasi penyewa
- Fitur yang didukung dan batasan
- Didukung Wilayah AWS
- Pertimbangan-pertimbangan
- Harga
- Mode isolasi
- Mengaktifkan isolasi penyewa untuk Fungsi Acara Lambda
- Memanggil Fungsi Acara Lambda dengan isolasi penyewa
- Mengakses pengenal penyewa dalam kode Fungsi Peristiwa Lambda
- Memantau Fungsi Acara Lambda dengan isolasi penyewa
- Memecahkan masalah isolasi penyewa untuk Fungsi Acara Lambda

Kapan menggunakan mode isolasi penyewa

Gunakan mode isolasi penyewa saat Anda perlu melayani beberapa pengguna akhir atau penyewa menggunakan satu Fungsi Peristiwa Lambda, sambil memastikan bahwa lingkungan eksekusi yang digunakan untuk memproses pemanggilan untuk penyewa individu tetap terisolasi satu sama lain. Isolasi lingkungan eksekusi yang ketat ini diperlukan untuk aplikasi multi-penyewa yang:

- Jalankan kode yang disediakan pengguna akhir: Mempertahankan lingkungan eksekusi terpisah untuk penyewa individu dapat membatasi dampak mengeksekusi kode yang disediakan pengguna yang mungkin salah atau berbahaya.
- Memproses data khusus penyewa: Mempertahankan lingkungan eksekusi terpisah untuk penyewa individu dapat mencegah paparan data spesifik penyewa yang sensitif ke penyewa lain.

Beberapa permintaan pemanggilan dari penyewa yang sama dapat menggunakan kembali lingkungan eksekusi fungsi yang sama, mengurangi start dingin dan meningkatkan waktu respons untuk aplikasi sensitif latensi.

Fitur yang didukung dan batasan

Mode isolasi penyewa tidak didukung dengan Fungsi Peristiwa yang menggunakan URL fungsi, konkurensi yang disediakan, atau SnapStart, atau dengan Fungsi Peristiwa yang berjalan di Instans Terkelola Lambda. Anda dapat mengirim permintaan ke fungsi terisolasi penyewa menggunakan

pemanggilan sinkron, pemanggilanasinkron, atau dengan menggunakan Amazon API Gateway sebagai pemicu peristiwa.

Didukung Wilayah AWS

Mode isolasi penyewa didukung di semua Wilayah komersial kecuali Asia Pasifik (Selandia Baru).

Pertimbangan-pertimbangan

Saat menggunakan isolasi penyewa dengan Fungsi Acara Lambda Anda, ingatlah hal berikut:

- Konfigurasi yang tidak dapat diubah: Isolasi penyewa adalah properti fungsi yang tidak dapat diubah. Itu hanya dapat diaktifkan saat membuat fungsi.
- Parameter tenant-id yang diperlukan: Fungsi Acara menggunakan mode isolasi penyewa harus dipanggil dengan parameter. `tenant-id` Menghilangkan parameter ini menyebabkan pemanggilan fungsi gagal.
- Peran eksekusi berlaku untuk semua penyewa: Pemanggilan dari semua penyewa menggunakan izin yang ditentukan dalam peran eksekusi Fungsi Peristiwa Lambda Anda.
- Konkurensi: Tidak ada perubahan pada konkurensi atau perilaku penskalaan Fungsi Acara Anda saat menggunakan isolasi penyewa. Lambda memberlakukan batas 2.500 lingkungan eksekusi terisolasi penyewa (aktif atau idle) untuk setiap 1.000 eksekusi bersamaan yang dikonfigurasi untuk Fungsi Peristiwa Lambda Anda.

Harga

Anda dikenakan biaya saat Lambda membuat lingkungan eksekusi baru yang terisolasi oleh penyewa. Harganya tergantung pada jumlah memori yang Anda alokasikan ke Fungsi Acara dan arsitektur CPU yang Anda gunakan. Untuk informasi selengkapnya, lihat harga AWS Lambda.

Mode isolasi

Tabel berikut menguraikan perbedaan antara Fungsi Peristiwa Lambda dengan dan tanpa isolasi penyewa.

Fitur	Dengan isolasi penyewa	Tanpa isolasi penyewa
Jenis isolasi	Tenant-level isolasi	Function-level isolasi

Fitur	Dengan isolasi penyewa	Tanpa isolasi penyewa
Penggunaan kembali lingkungan	Lingkungan eksekusi tidak pernah digunakan kembali di berbagai penyewa	Lingkungan eksekusi dapat digunakan kembali di seluruh pemanggilan dari fungsi yang sama
Isolasi data	Data dari penyewa lain tidak dapat diakses	Data dari pemanggilan sebelumnya dari versi fungsi yang sama mungkin dapat diakses
Dingin dimulai	Lebih dingin dimulai karena lingkungan khusus penyewa	Lebih sedikit dingin dimulai karena penggunaan kembali lingkungan
Harga	Biaya tambahan selain harga Lambda standar	Harga Lambda Standar

Mengaktifkan isolasi penyewa untuk Fungsi Acara Lambda

Untuk mengaktifkan mode isolasi penyewa, buat Fungsi Acara Lambda baru. Anda tidak dapat mengaktifkan isolasi penyewa pada Fungsi Acara yang ada.

Topik

- Mengaktifkan isolasi penyewa (konsol)
- Mengaktifkan isolasi penyewa (AWS CLI)
- Mengaktifkan isolasi penyewa (API)
- Mengaktifkan isolasi penyewa (CloudFormation)

Mengaktifkan isolasi penyewa (konsol)

Untuk membuat Fungsi Peristiwa Lambda menggunakan konsol

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih Buat Fungsi Acara.
3. Pilih Penulis dari awal.
4. Di panel Informasi dasar, untuk nama Fungsi, masukkan**`image-analysis`**.
5. Untuk Runtime, pilih salah satu runtime Lambda yang didukung.
6. Di bawah konfigurasi tambahan, mode isolasi penyewa, pilih Aktifkan.
7. Tinjau pengaturan Anda, dan pilih Create Event Function.

Mengaktifkan isolasi penyewa (AWS CLI)

Buat Fungsi Acara dengan isolasi penyewa

Saat membuat Fungsi Acara baru menggunakan CLI, tambahkan `--tenancy-config '{"TenantIsolationMode": "PER_TENANT"}'` opsi ke permintaan fungsi buat Anda. Contoh:

```
aws lambda create-function \
    --function-name image-analysis \
    --runtime nodejs24.x \
    --zip-file fileb://image-analysis-function.zip \
    --handler image-analysis-function.handler \
    --role arn:aws:iam:123456789012:role/execution-role \
```

```
    --tenancy-config '{"TenantIsolationMode": "PER_TENANT"}'
```

Mengaktifkan isolasi penyewa (API)

Untuk mengaktifkan isolasi penyewa menggunakan Lambda API

1. Buat Fungsi Acara baru dengan isolasi penyewa diaktifkan dengan menggunakan aksi CreateFunctionAPI dengan `TenancyConfig` parameter.
2. Konfirmasikan bahwa isolasi penyewa diaktifkan untuk Fungsi Acara dengan menggunakan GetFunctionConfigurationtindakan. Jika respons menunjukkan `TenantIsolationMode` ituPER_TENANT, maka isolasi penyewa diaktifkan untuk Fungsi Peristiwa:

   ```
   "TenancyConfig": {
           "TenantIsolationMode": "PER_TENANT"
       }
   ```

Panggil versi fungsi dengan tindakan Invoke. Untuk informasi selengkapnya, lihat Memanggil Fungsi Acara Lambda dengan isolasi penyewa.

Mengaktifkan isolasi penyewa (CloudFormation)

CloudFormation Template berikut membuat Fungsi Peristiwa Lambda baru dengan isolasi penyewa diaktifkan:

```
MyLambdaFunction:
    Type: AWS::Lambda::Function
    Properties:
      FunctionName: my-sample-python-lambda
      Runtime: python3.14
      Role: !GetAtt LambdaExecutionRole.Arn
      Handler: index.lambda_handler
      TenancyConfig:
        TenantIsolationMode: PER_TENANT
      Code:
        ZipFile: |
          import json

          def lambda_handler(event, context):
              return {
                  'statusCode': 200,
```

```
                'body': json.dumps(f'Hello from Lambda! Tenant-ID:
{context.tenant_id}')
            }
    Timeout: 10
    MemorySize: 128
```

Memanggil Fungsi Acara Lambda dengan isolasi penyewa

Saat menjalankan Fungsi Peristiwa yang mengaktifkan isolasi penyewa, Anda harus memberikan parameter. `tenant-id` Parameter ini memastikan bahwa pemanggilan Fungsi Acara Anda diproses dalam lingkungan eksekusi yang didedikasikan untuk penyewa tertentu.

Memanggil Fungsi Acara dengan isolasi penyewa (AWS CLI)

Pemanggilan sinkron

Untuk pemanggilan sinkron, tambahkan `--tenant-id` parameter ke perintah Invoke Anda:

```
aws lambda invoke \
    --function-name image-analysis \
    --tenant-id blue \
    response.json
```

Pemanggilan asinkron

Untuk pemanggilan asinkron, sertakan parameter dan parameter: `--tenant-id` `--invocation-type Event`

```
aws lambda invoke \
    --function-name image-analysis \
    --tenant-id blue \
    --invocation-type Event \
    response.json
```

Memanggil Fungsi Acara dengan isolasi penyewa (API)

Saat menggunakan tindakan Invoke API secara langsung, sertakan pengenal penyewa menggunakan `X-Amzn-Tenant-Id` parameter dalam permintaan Anda.

Contoh permintaan API

```
POST /2015-03-31/functions/image-analysis/invocations HTTP/1.1
Host: lambda.us-east-1.amazonaws.com
Content-Type: application/json
Authorization: AWS4-HMAC-SHA256 Credential=...
X-Amz-Tenant-Id: blue

{
```

```
    "key1": "value1",
    "key2": "value2"
}
```

Memanggil Fungsi Acara dengan isolasi penyewa (API Gateway)

Saat menggunakan API REST API Gateway untuk memicu Fungsi Peristiwa Lambda yang terisolasi penyewa, Anda harus mengonfigurasi API Gateway untuk memetakan properti permintaan klien ke `X-Amz-Tenant-Id` header yang diharapkan Lambda. API Gateway menggunakan aksi Invoke API Lambda, yang mengharuskan ID penyewa diteruskan menggunakan header HTTP. `X-Amz-Tenant-Id` Anda dapat mengonfigurasi API Gateway untuk menyuntikkan header HTTP ini ke permintaan pemanggilan Lambda dengan nilai yang diperoleh dari properti permintaan klien seperti header HTTP, parameter kueri, atau parameter jalur. Anda harus terlebih dahulu memetakan properti permintaan klien sebelum Anda dapat mengganti `X-Amz-Tenant-Id` header.

> **Note**
>
> Anda tidak dapat menggunakan API HTTP untuk memanggil Fungsi Peristiwa Lambda yang terisolasi penyewa karena header tidak dapat di-override. `X-Amz-Tenant-Id`

Menggunakan header permintaan

Konfigurasikan integrasi API Gateway Anda untuk memetakan header kustom dari permintaan klien ke `X-Amz-Tenant-Id` header. Contoh berikut menunjukkan permintaan klien dengan `x-tenant-id` header:

```
POST /api/process HTTP/1.1
Host: your-api-id.execute-api.us-east-1.amazonaws.com
Content-Type: application/json
x-tenant-id: blue

{
    "data": "sample payload"
}
```

Dalam konfigurasi metode API Gateway Anda, Anda harus:

1. Aktifkan parameter header permintaan klien (misalnya,`method.request.header.x-tenant-id`)

2. Petakan header klien ke header integrasi Lambda menggunakan `integration.request.header.X-Amz-Tenant-Id`

Menggunakan parameter kueri

Demikian pula, Anda dapat memetakan parameter kueri ke `X-Amz-Tenant-Id` header:

```
GET /api/process?tenant-id=blue&data=sample HTTP/1.1
Host: your-api-id.execute-api.us-east-1.amazonaws.com
```

Konfigurasikan metode untuk mengaktifkan parameter kueri dan memetakannya ke header integrasi.

Memanggil Fungsi Acara dengan isolasi penyewa (SDK)

Saat menggunakan AWS SDK untuk memanggil Fungsi Peristiwa yang terisolasi penyewa, sertakan pengenal penyewa dalam permintaan pemanggilan Anda.

Python

```
import boto3
import json

lambda_client = boto3.client('lambda')

response = lambda_client.invoke(
    FunctionName='image-analysis',
    TenantId='blue',
    Payload=json.dumps({
        'key1': 'value1',
        'key2': 'value2'
    })
)

result = json.loads(response['Payload'].read())
```

Node.js

```
const AWS = require('aws-sdk');
const lambda = new AWS.Lambda();

const params = {
```

```
    FunctionName: 'image-analysis',
    TenantId: 'blue',
    Payload: JSON.stringify({
        key1: 'value1',
        key2: 'value2'
    })
};

lambda.invoke(params, (err, data) => {
    if (err) {
        console.error(err);
    } else {
        const result = JSON.parse(data.Payload);
        console.log(result);
    }
});
```

Java

```
import software.amazon.awssdk.services.lambda.LambdaClient;
import software.amazon.awssdk.services.lambda.model.InvokeRequest;
import software.amazon.awssdk.services.lambda.model.InvokeResponse;
import software.amazon.awssdk.core.SdkBytes;

public class TenantIsolationExample {

    public static void main(String[] args) {
        LambdaClient lambdaClient = LambdaClient.create();

        String payload = "{\"key1\": \"value1\", \"key2\": \"value2\"}";

        InvokeRequest request = InvokeRequest.builder()
                .functionName("image-analysis")
                .tenantId("blue")
                .payload(SdkBytes.fromUtf8String(payload))
                .build();

        InvokeResponse response = lambdaClient.invoke(request);
    }
}
```

Mengakses pengenal penyewa dalam kode Fungsi Peristiwa Lambda

Ketika Fungsi Peristiwa Lambda Anda mengaktifkan isolasi penyewa, pengenal penyewa yang digunakan untuk memanggil Fungsi Acara Anda tersedia dalam objek konteks yang diteruskan ke handler Fungsi Peristiwa Anda. Anda dapat menggunakan pengenal ini untuk mengimplementasikan logika, pemantauan, dan kemampuan debugging khusus penyewa.

Topik

- Mengakses pengenal penyewa
- Pola penggunaan umum
- Pemantauan dan debugging

Mengakses pengenal penyewa

Pengenal penyewa tersedia melalui `tenantId` properti objek konteks. Perhatikan bahwa properti ini tersedia selama fase pemanggilan, bukan selama fase inisialisasi.

Python

```
def lambda_handler(event, context):
    tenant_id = context.tenant_id
    print(f"Processing request for tenant: {tenant_id}")

    # Implement tenant-specific logic
    if tenant_id == "blue":
        return process_blue_tenant(event)
    elif tenant_id == "green":
        return process_green_tenant(event)
    else:
        return process_default_tenant(event)
```

Node.js

```
exports.handler = async (event, context) => {
    const tenantId = context.tenantId;
    console.log(`Processing request for tenant: ${tenantId}`);

    // Implement tenant-specific logic
    switch (tenantId) {
        case 'blue':
```

```
            return processBlueTenant(event);
        case 'green':
            return processGreenTenant(event);
        default:
            return processDefaultTenant(event);
    }
};
```

Java

```
public class TenantHandler implements RequestHandler<Map<String, Object>, String> {

    @Override
    public String handleRequest(Map<String, Object> event, Context context) {
        String tenantId = context.getTenantId();
        System.out.println("Processing request for tenant: " + tenantId);

        // Implement tenant-specific logic
        switch (tenantId) {
            case "blue":
                return processBlueTenant(event);
            case "green":
                return processGreenTenant(event);
            default:
                return processDefaultTenant(event);
        }
    }
}
```

Pola penggunaan umum

Berikut adalah cara umum untuk menggunakan pengenal penyewa dalam kode Fungsi Acara Anda:

Tenant-specific konfigurasi

Gunakan ID penyewa untuk memuat konfigurasi atau pengaturan khusus penyewa:

```
def lambda_handler(event, context):
    tenant_id = context.tenant_id

    # Load tenant-specific configuration
    config = load_tenant_config(tenant_id)
```

```
    database_url = config['database_url']
    api_key = config['api_key']

    # Process with tenant-specific settings
    return process_request(event, database_url, api_key)
```

Tenant-specific akses data

Gunakan ID penyewa untuk memastikan isolasi data dan kontrol akses:

```
import boto3

def lambda_handler(event, context):
    tenant_id = context.tenant_id

    # Ensure data access is scoped to the tenant
    dynamodb = boto3.resource('dynamodb')
    table = dynamodb.Table('user_data')

    user_id = event.get('userId')

    response = table.get_item(
        Key={
            'tenant_id': tenant_id,
            'user_id': user_id
        }
    )

    return process_results(response.get('Item'), tenant_id)
```

Pemantauan dan debugging

Pengenal penyewa secara otomatis disertakan dalam log Lambda saat Anda mengaktifkan logging JSON, sehingga memudahkan untuk memantau dan men-debug masalah khusus penyewa. Anda juga dapat menggunakan ID penyewa untuk metrik dan penelusuran khusus.

Example Metrik kustom dengan ID penyewa

Contoh berikut menunjukkan cara menggunakan ID penyewa untuk membuat CloudWatch metrik khusus penyewa untuk memantau pola penggunaan dan kinerja oleh penyewa:

```
import boto3
```

```
def lambda_handler(event, context):
    tenant_id = context.tenant_id
    cloudwatch = boto3.client('cloudwatch')

    # Record tenant-specific metrics
    cloudwatch.put_metric_data(
        Namespace='MyApp/TenantMetrics',
        MetricData=[
            {
                'MetricName': 'RequestCount',
                'Dimensions': [
                    {
                        'Name': 'TenantId',
                        'Value': tenant_id
                    }
                ],
                'Value': 1,
                'Unit': 'Count'
            }
        ]
    )

    return process_request(event, tenant_id)
```

Memantau Fungsi Acara Lambda dengan isolasi penyewa

Anda dapat memantau fungsi Lambda yang terisolasi penyewa menggunakan CloudWatch AWS X-Ray Amazon, dan dengan mengakses data telemetri real-time untuk ekstensi menggunakan API Telemetri.

Memahami logging untuk mode terisolasi penyewa

Untuk fungsi yang menggunakan isolasi penyewa, Lambda secara otomatis menyertakan pengenal penyewa dalam log fungsi saat Anda mengaktifkan logging JSON, sehingga memudahkan untuk memantau dan men-debug masalah khusus penyewa. Lambda membuat aliran CloudWatch log terpisah untuk setiap lingkungan eksekusi. Anda dapat menggunakan Wawasan CloudWatch Log untuk menemukan aliran log milik penyewa tertentu dengan memfilter berdasarkan pengenal penyewa:

```
fields @logStream, @message
| filter tenantId=='BlueTenant' or record.tenantId=='BlueTenant'
| stats count() as logCount by @logStream
| sort @timestamp desc
```

Anda juga dapat menggunakan parameter ini untuk mengambil log khusus penyewa di semua aliran log:

```
fields @message
| filter tenantId=='BlueTenant' or record.tenantId=='BlueTenant'
| limit 1000
```

`tenantId`Properti disertakan untuk acara platform (seperti `platform.start` dan`platform.report`) dan log kustom yang Anda cetak dalam kode fungsi Anda, seperti yang ditunjukkan di bawah ini:

```
{
    "time": "2025-10-13T19:48:06.990Z",
    "type": "platform.start",
    "record": {
        "requestId": "a0f40320-b43c-44b3-91bf-d5b5240a1bed",
        "functionArn": "arn:aws:lambda:us-east-1:xxxxxx:function:multitenant-
function-1",
        "version": "$LATEST",
        "tenantId": "BlueTenant""
```

```
    }
}
{
    "timestamp": "2025-10-13T19:48:06.992Z",
    "level": "INFO",
    "requestId": "a0f40320-b43c-44b3-91bf-d5b5240a1bed",
    "tenantId": "BlueTenant"",
    "message": "custom log line1"
}
{
    "timestamp": "2025-10-13T19:48:07.022Z",
    "level": "WARN",
    "requestId": "a0f40320-b43c-44b3-91bf-d5b5240a1bed",
    "tenantId": "BlueTenant"",
    "message": "custom log line2"
}
```

Memecahkan masalah isolasi penyewa untuk Fungsi Acara Lambda

Halaman ini membahas masalah umum yang terjadi saat menggunakan isolasi penyewa untuk AWS Lambda.

InvalidParameterValueException

Kesalahan: Konfigurasi ID penyewa tidak ditentukan atau diteruskan ke fungsi saat isolasi penyewa tidak diaktifkan.

Penyebab umum

Kesalahan ini terjadi saat menjalankan fungsi terisolasi penyewa tanpa ID penyewa, atau menjalankan fungsi non-tenant-isolated dengan ID penyewa.

Resolusi

Tambahkan ID penyewa jika fungsi mengaktifkan isolasi penyewa, atau hapus ID penyewa jika fungsi tidak mengaktifkan isolasi penyewa.

TooManyRequestsException

Kesalahan: Tingkat terlampaui

Penyebab umum

Selain pembatasan tarif berdasarkan eksekusi bersamaan maksimum dan laju penskalaan fungsi, Lambda membatasi jumlah maksimum lingkungan eksekusi sadar penyewa (aktif atau idle) yang dapat eksis sekaligus menjadi 2.500 untuk setiap 1.000 eksekusi fungsi Anda secara bersamaan.

Resolusi

Untuk memperbaiki masalah ini, Anda dapat menurunkan tingkat permintaan pemanggilan dengan pengidentifikasi penyewa unik dibuat, menerapkan percobaan ulang dengan backoff dan jitter, atau meminta peningkatan batas konkurensi fungsi.

Instans Terkelola Lambda

Dengan Instans Terkelola Lambda, Anda dapat menjalankan Lambda Event Functions pada instans Amazon EC2 generasi saat ini, termasuk Graviton4, instans yang dioptimalkan untuk jaringan, dan

opsi komputasi khusus lainnya. Anda tidak perlu mengelola siklus hidup instans, sistem operasi, dan kebijakan runtime bahasa, perutean, penyeimbangan beban, atau penskalaan. Dengan Instans Terkelola Lambda, Anda mendapatkan keuntungan dari keunggulan harga EC2, termasuk Paket Tabungan EC2 dan Instans Cadangan. Instans Terkelola Lambda saat ini hanya mendukung Lambda Event Functions.

Untuk daftar jenis instans yang didukung, buka AWS Lambda Harga halaman dan pilih Wilayah AWS.

Kemampuan Utama

Instans Terkelola Lambda menyediakan kemampuan berikut:

- Pilih contoh yang sesuai - Pilih contoh yang sesuai berdasarkan persyaratan kinerja dan biaya, termasuk akses ke CPU terbaru seperti Graviton4, rasio Memori-CPU yang dapat dikonfigurasi, dan jaringan bandwidth tinggi.
- Penyediaan otomatis - AWS secara otomatis menyediakan contoh yang sesuai dan memutar lingkungan eksekusi fungsi.
- Penskalaan dinamis - Instans berskala secara dinamis berdasarkan pola lalu lintas fungsi Anda.
- Pengalaman yang dikelola sepenuhnya - AWS menangani manajemen infrastruktur, penskalaan, tambalan, dan perutean, dengan integrasi sumber peristiwa ekstensif yang sama dengan yang Anda kenal.

Kapan menggunakan Instans Terkelola Lambda

Pertimbangkan Instans Terkelola Lambda untuk kasus penggunaan berikut:

- Beban kerja yang dapat diprediksi volume tinggi - Ideal untuk beban kerja kondisi mapan tanpa lonjakan lalu lintas yang tidak terduga. Lambda Managed Instances dapat diskalakan untuk menangani penggandaan lalu lintas dalam waktu lima menit secara default.
- Performance-critical aplikasi - Akses ke CPU terbaru, berbagai rasio Memori-CPU, dan throughput jaringan yang tinggi
- Persyaratan peraturan - Kebutuhan tata kelola granular dengan kontrol atas VPC dan penempatan instans
- Berbagai aplikasi - Event-driven aplikasi, media/data pemrosesan, aplikasi web, dan beban kerja lama yang bermigrasi ke tanpa server

Cara Kerjanya

Instans Terkelola Lambda menggunakan penyedia kapasitas sebagai dasar untuk menjalankan fungsi Anda:

1. Buat penyedia kapasitas - Tentukan di mana fungsi Anda berjalan dengan menentukan konfigurasi VPC dan opsional, persyaratan instance, dan konfigurasi penskalaan
2. Buat fungsi Anda - Buat fungsi Lambda seperti biasa dan lampirkan ke penyedia kapasitas
3. Publikasikan versi fungsi - Versi fungsi menjadi aktif pada instance penyedia kapasitas setelah diterbitkan

Saat Anda menerbitkan versi fungsi dengan penyedia kapasitas, Lambda meluncurkan Instans Terkelola di akun Anda. Ini meluncurkan tiga instance secara default untuk ketahanan AZ dan memulai tiga lingkungan eksekusi sebelum menandai versi fungsi Anda AKTIF. Jika Anda melampirkan fungsi ke penyedia kapasitas yang sudah menjalankan fungsi lain, Lambda mungkin tidak memutar instans baru jika instans yang tersedia sudah memiliki kapasitas untuk mengakomodasi lingkungan eksekusi fungsi baru.

Model konkurensi

Instans Terkelola Lambda mendukung pemanggilan multi-serentak, di mana satu lingkungan eksekusi dapat menangani beberapa pemanggilan secara bersamaan. Ini berbeda dari tipe komputasi Lambda (default), yang menyediakan model konkurensi tunggal di mana satu lingkungan eksekusi dapat menjalankan maksimal satu pemanggilan pada satu waktu. Multi-concurrency menghasilkan pemanfaatan yang lebih baik dari instans EC2 yang mendasarinya dan sangat bermanfaat untuk IO-heavy aplikasi seperti layanan web atau pekerjaan batch. Perubahan dalam model eksekusi ini berarti bahwa keamanan utas, manajemen status, dan isolasi konteks harus ditangani secara berbeda tergantung pada runtime.

Sewa dan isolasi

Jenis komputasi Lambda (default) adalah multi-tenant, memanfaatkan teknologi MicroVM Firecracker untuk memberikan isolasi antara lingkungan eksekusi yang berjalan pada armada Lambda bersama. Instans Terkelola Lambda berjalan di akun Anda, menyediakan opsi perangkat keras dan harga EC2 terbaru. Instans Terkelola menggunakan wadah yang berjalan pada instans EC2 Nitro untuk memberikan isolasi daripada Firecracker. Penyedia kapasitas berfungsi sebagai batas keamanan untuk fungsi Lambda. Fungsi dijalankan dalam wadah dalam instance.

Memahami instans terkelola

Fungsi Instans Terkelola Lambda berjalan pada Instans terkelola EC2 di akun Anda. Instans ini sepenuhnya dikelola oleh Lambda, yang berarti Anda memiliki izin terbatas pada instans tersebut dibandingkan dengan instans EC2 standar. Anda dapat mengidentifikasi Instans Terkelola Lambda di akun Anda dengan:

- Kehadiran `Operator` bidang di EC2 `DescribeInstances` keluaran
- `aws:lambda:capacity-provider` tag pada instance

Anda tidak dapat melakukan operasi EC2 standar secara langsung pada instans ini, seperti menghentikannya secara manual. Untuk menghancurkan instans terkelola, hapus penyedia kapasitas terkait. Lambda kemudian mengakhiri instance sebagai bagian dari proses penghapusan penyedia kapasitas.

Instans terkelola disembunyikan dari tampilan konsol EC2 dan operasi daftar API secara default. Anda dapat menyesuaikan visibilitas menggunakan pengaturan visibilitas sumber daya terkelola. Instans terkelola tetap beroperasi penuh dan dapat ditagih di AWS akun.

Harga

Instans Terkelola Lambda menggunakan EC2-based harga dengan biaya manajemen 15% di atas biaya instans EC2. Model penetapan harga ini mendukung Paket Tabungan EC2, Instans Cadangan, dan diskon harga lainnya yang diterapkan pada penggunaan EC2 Anda. Lihat halaman harga untuk detail tambahan: https://aws.amazon.com/lambda/pricing/

Untuk membandingkan biaya menjalankan fungsi Anda pada default Lambda, Instans Terkelola Lambda, dan Amazon EC2 yang dikelola sendiri, gunakan Kalkulator harga Instans Terkelola Lambda.

Penting: Diskon harga EC2 hanya berlaku untuk komputasi EC2 yang mendasarinya, bukan untuk biaya manajemen.

Bagaimana Instans Terkelola Lambda berbeda dari tipe komputasi Lambda (default)

Lambda Managed Instances mengubah cara Lambda memproses permintaan dibandingkan dengan Lambda (default).

Perbedaan utama:

	Lambda (default)	Instans Terkelola Lambda
Model konkurensi	Model konkurensi tunggal di mana satu lingkungan eksekusi dapat mendukung maksimal satu pemanggilan pada satu waktu	Multi-concurrent pemanggilan di mana satu lingkungan eksekusi dapat menangani beberapa pemanggilan secara bersamaan, meningkatkan throughput terutama untuk IO-heavy aplikasi
Sewa dan isolasi	Multi-tenant, menggunakan teknologi MicroVM Firecracker untuk memberikan isolasi antara lingkungan eksekusi yang berjalan pada armada Lambda bersama	Jalankan di akun Anda, menggunakan EC2 Nitro untuk memberikan isolasi. Penyedia kapasitas berfungsi sebagai batas keamanan, dengan fungsi yang dieksekusi dalam wadah dalam instance
Model penentuan harga	Per-request harga durasi	Instance-based harga dengan model harga EC2, termasuk On-Demand dan Instans Cadangan, dan opsi penghematan seperti Compute Savings Plans
Perilaku penskalaan	Menskalakan ketika tidak ada lingkungan eksekusi bebas untuk menangani pemanggilan masuk (start dingin). Skala ke nol tanpa lalu lintas	Menskalakan secara asinkron berdasarkan pemanfaatan sumber daya CPU saja, tanpa cold start. Menskalakan ke lingkungan eksekusi minimum yang dikonfigu rasi tanpa lalu lintas
Paling cocok untuk	Berfungsi dengan lalu lintas padat yang dapat menangani beberapa waktu mulai dingin, atau aplikasi tanpa beban berkelanjutan yang mendapat manfaat dari skala hingga nol	Lalu lintas yang dapat diprediksi volume tinggi berfungsi saat Anda menginginkan fleksibilitas, paket harga, dan opsi perangkat keras EC2

Langkah selanjutnya

- Pelajari tentang konsep inti untuk Instans Terkelola Lambda
- Memulai dengan Instans Terkelola Lambda
- Pelajari tentang penyedia kapasitas untuk Instans Terkelola Lambda
- Memahami penskalaan untuk Instans Terkelola Lambda
- Pelajari tentang menyebarkan dan memperbarui fungsi Lambda Managed Instances
- Tinjauan runtime yang didukung
- Tinjau panduan khusus runtime untuk Java, Node.js, dan Python
- Konfigurasikan Konektivitas VPC untuk penyedia kapasitas Anda
- Memahami keamanan dan izin untuk Instans Terkelola Lambda
- Tinjauan kuota dan batasan
- Memecahkan Masalah Instans Terkelola Lambda

Memulai Instans Terkelola Lambda

Membuat fungsi Instans Terkelola Lambda (konsol)

Anda dapat menggunakan konsol Lambda untuk membuat fungsi Instans Terkelola yang berjalan di instans Amazon EC2 yang dikelola oleh penyedia kapasitas.

Penting: Sebelum membuat fungsi Instans Terkelola, Anda harus terlebih dahulu membuat penyedia kapasitas. Fungsi Peristiwa Lambda ini memerlukan penyedia kapasitas untuk menentukan infrastruktur Amazon EC2 yang menjalankan fungsi Anda.

Untuk membuat fungsi Instans Terkelola Lambda (konsol)

1. Buka Konsol Lambda.
2. Pilih Penyedia kapasitas dari panel navigasi kiri.
3. Pilih Buat penyedia kapasitas.
4. Di bagian Pengaturan penyedia kapasitas, masukkan nama untuk penyedia kapasitas Anda.
5. Pilih VPC dan izin untuk penyedia kapasitas Anda. Anda dapat menggunakan yang sudah ada atau membuat yang baru. Untuk informasi tentang membuat peran operator yang diperlukan, lihat Peran Operator Lambda untuk Instans Terkelola Lambda.
6. Perluas Pengaturan lanjutan.

7. Tentukan persyaratan Instance Anda dengan memilih arsitektur prosesor dan jenis instans.
8. Di bawah Penskalaan otomatis, tentukan jumlah maksimum vCPU EC2 untuk penyedia kapasitas Anda. Anda juga dapat memilih mode penskalaan instans Manual untuk mengatur nilai penskalaan Anda sendiri untuk kontrol yang tepat.
9. Pilih Buat penyedia kapasitas untuk membuat yang baru.
10. Selanjutnya, pilih Create function.
11. Pilih Penulis dari awal.
12. Di panel Informasi dasar, berikan nama Fungsi.
13. Untuk Runtime, pilih salah satu Runtime yang didukung.
14. Pilih Arsitektur untuk fungsi Anda (sama seperti yang Anda pilih untuk penyedia kapasitas). Secara default, x86_64.
15. Di bawah Izin, pastikan Anda memiliki izin untuk peran Eksekusi yang dipilih. Jika tidak, Anda dapat membuat peran baru.
16. Di bawah Konfigurasi tambahan, pilih tipe Komputasi sebagai Instans Terkelola Lambda.
17. Penyedia kapasitas ARN dari penyedia kapasitas yang Anda buat pada langkah sebelumnya harus dipilih sebelumnya.
18. Pilih Ukuran memori dan Memori lingkungan eksekusi (GiB) per rasio vCPU.
19. Pilih Buat fungsi.

Fungsi Instans Terkelola Lambda Anda dibuat dan menyediakan kapasitas pada penyedia kapasitas yang Anda tentukan. Pembuatan fungsi biasanya memakan waktu beberapa menit. Setelah selesai, Anda dapat mengedit kode fungsi Anda dan menjalankan tes pertama Anda.

Membuat fungsi Instans Terkelola Lambda (AWS CLI)

Prasyarat

Sebelum Anda mulai, pastikan Anda memiliki yang berikut:

- AWS CLI - Instal dan konfigurasikan CLI AWS . Untuk informasi selengkapnya, lihat Menginstal atau memperbarui AWS CLI versi terbaru.
- Izin IAM — Pengguna atau peran IAM Anda harus memiliki izin untuk membuat fungsi Lambda, penyedia kapasitas, dan meneruskan peran IAM. Perhatikan bahwa Anda juga perlu `iam:CreateServiceLinkedRole` jika ini pertama kalinya membuat penyedia kapasitas di akun atau jika Peran Tertaut Layanan (SLR) telah dihapus.

Langkah 1: Buat peran IAM yang diperlukan

Instans Terkelola Lambda memerlukan dua peran IAM: peran eksekusi untuk fungsi Anda dan peran operator untuk penyedia kapasitas. Peran operator memungkinkan Lambda meluncurkan, menghentikan, dan memantau instans Amazon EC2 atas nama Anda. Peran eksekusi fungsi memberikan izin fungsi untuk mengakses AWS layanan dan sumber daya lainnya.

Untuk membuat peran eksekusi Lambda

1. Buat dokumen kebijakan kepercayaan yang memungkinkan Lambda untuk mengambil peran:

```
cat > lambda-trust-policy.json << 'EOF'
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "lambda.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
EOF
```

2. Buat peran eksekusi:

```
aws iam create-role \
  --role-name MyLambdaExecutionRole \
  --assume-role-policy-document file://lambda-trust-policy.json
```

3. Lampirkan kebijakan eksekusi dasar:

```
aws iam attach-role-policy \
  --role-name MyLambdaExecutionRole \
  --policy-arn arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole
```

Untuk membuat peran operator penyedia kapasitas

1. Buat dokumen kebijakan kepercayaan yang memungkinkan Lambda untuk mengambil peran operator:

```
cat > operator-trust-policy.json << 'EOF'
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "lambda.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
EOF
```

2. Buat peran operator:

```
aws iam create-role \
  --role-name MyCapacityProviderOperatorRole \
  --assume-role-policy-document file://operator-trust-policy.json
```

3. Lampirkan kebijakan izin EC2 yang diperlukan:

```
aws iam attach-role-policy \
  --role-name MyCapacityProviderOperatorRole \
  --policy-arn arn:aws:iam::aws:policy/AWSLambdaManagedEC2ResourceOperator
```

Langkah 2: Siapkan sumber daya VPC

Instans Terkelola Lambda berjalan di VPC Anda dan memerlukan subnet dan grup keamanan.

Untuk membuat sumber daya VPC

1. Membuat VPC:

```
VPC_ID=$(aws ec2 create-vpc \
  --cidr-block 10.0.0.0/16 \
  --query 'Vpc.VpcId' \
```

```
  --output text)
```

2. Buat subnet:

```
SUBNET_ID=$(aws ec2 create-subnet \
  --vpc-id $VPC_ID \
  --cidr-block 10.0.1.0/24 \
  --query 'Subnet.SubnetId' \
  --output text)
```

3. Buat grup keamanan:

```
SECURITY_GROUP_ID=$(aws ec2 create-security-group \
  --group-name my-capacity-provider-sg \
  --description "Security group for Lambda Managed Instances" \
  --vpc-id $VPC_ID \
  --query 'GroupId' \
  --output text)
```

Catatan: Fungsi Instans Terkelola Lambda Anda memerlukan konfigurasi VPC untuk mengakses sumber daya di luar VPC dan untuk mengirimkan data telemetri ke Log dan. CloudWatch X-Ray Untuk detail konfigurasi, lihat Jaringan untuk Instans Terkelola Lambda.

Langkah 3: Buat penyedia kapasitas

Penyedia kapasitas mengelola instans EC2 yang menjalankan fungsi Lambda Anda.

Untuk membuat penyedia kapasitas

```
ACCOUNT_ID=$(aws sts get-caller-identity --query Account --output text)

aws lambda create-capacity-provider \
  --capacity-provider-name my-capacity-provider \
  --vpc-config SubnetIds=[$SUBNET_ID],SecurityGroupIds=[$SECURITY_GROUP_ID] \
  --permissions-config CapacityProviderOperatorRoleArn=arn:aws:iam::${ACCOUNT_ID}:role/
MyCapacityProviderOperatorRole \
  --instance-requirements Architectures=[x86_64] \
  --capacity-provider-scaling-config MaxVCpuCount=30
```

Perintah ini membuat penyedia kapasitas dengan konfigurasi berikut:

- Konfigurasi VPC - Menentukan subnet dan grup keamanan untuk instans EC2

- Izin - Mendefinisikan peran IAM yang digunakan Lambda untuk mengelola instans EC2
- Persyaratan instans - Menentukan arsitektur x86_64
- Konfigurasi penskalaan - Menetapkan maksimum 30 vCPU untuk penyedia kapasitas

Langkah 4: Buat fungsi Lambda dengan kode sebaris

Untuk membuat fungsi dengan kode inline

1. Pertama, buat fungsi Python sederhana dan paketkan sebaris:

```
# Create a temporary directory for the function code
mkdir -p /tmp/my-lambda-function
cd /tmp/my-lambda-function

# Create a simple Python handler
cat > lambda_function.py << 'EOF'
import json

def lambda_handler(event, context):
    return {
        'statusCode': 200,
        'body': json.dumps({
            'message': 'Hello from Lambda Managed Instances!',
            'event': event
        })
    }
EOF

# Create a ZIP file
zip function.zip lambda_function.py
```

2. Buat fungsi Lambda menggunakan file ZIP sebaris:

```
ACCOUNT_ID=$(aws sts get-caller-identity --query Account --output text)
REGION=$(aws configure get region)

aws lambda create-function \
  --function-name my-managed-instance-function \
  --package-type Zip \
  --runtime python3.13 \
  --handler lambda_function.lambda_handler \
  --zip-file fileb:///tmp/my-lambda-function/function.zip \
```

```
  --role arn:aws:iam::${ACCOUNT_ID}:role/MyLambdaExecutionRole \
  --architectures x86_64 \
  --memory-size 2048 \
  --ephemeral-storage Size=512 \
  --capacity-provider-config
 LambdaManagedInstancesCapacityProviderConfig={CapacityProviderArn=arn:aws:lambda:
${REGION}:${ACCOUNT_ID}:capacity-provider:my-capacity-provider}
```

Fungsi ini dibuat dengan:

- Runtime - Python 3.13
- Handler — `lambda_handler` Fungsi di `lambda_function.py`
- Memori - 2048 MB
- Penyimpanan fana - 512 MB
- Penyedia kapasitas - Tautan ke penyedia kapasitas yang Anda buat

Langkah 5: Publikasikan versi fungsi

Untuk menjalankan fungsi di Instans Terkelola Lambda, Anda harus menerbitkan versi.

Untuk mempublikasikan versi fungsi

```
aws lambda publish-version \
  --function-name my-managed-instance-function
```

Perintah ini menerbitkan versi 1 dari fungsi Anda dan menyebarkannya ke penyedia kapasitas.

Langkah 6: Panggil fungsi Anda

Setelah menerbitkan, Anda dapat memanggil fungsi Anda.

Untuk memanggil fungsi Anda

```
aws lambda invoke \
  --function-name my-managed-instance-function:1 \
  --payload '{"name": "World"}' \
  response.json

# View the response
cat response.json
```

Fungsi ini berjalan pada instans EC2 yang dikelola oleh penyedia kapasitas Anda dan mengembalikan respons.

Bersihkan

Untuk menghindari biaya yang dikenakan, hapus sumber daya yang Anda buat:

1. Hapus fungsi:

```
aws lambda delete-function --function-name my-managed-instance-function
```

2. Hapus penyedia kapasitas:

```
aws lambda delete-capacity-provider --capacity-provider-name my-capacity-provider
```

3. Hapus sumber daya VPC:

```
aws ec2 delete-security-group --group-id $SECURITY_GROUP_ID
aws ec2 delete-subnet --subnet-id $SUBNET_ID
aws ec2 delete-vpc --vpc-id $VPC_ID
```

4. Hapus peran IAM:

```
aws iam detach-role-policy \
  --role-name MyLambdaExecutionRole \
  --policy-arn arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole
aws iam detach-role-policy \
  --role-name MyCapacityProviderOperatorRole \
  --policy-arn arn:aws:iam::aws:policy/AWSLambdaManagedEC2ResourceOperator

aws iam delete-role --role-name MyLambdaExecutionRole
aws iam delete-role --role-name MyCapacityProviderOperatorRole
```

Konsep inti

Lambda Managed Instances memperkenalkan beberapa konsep inti yang berbeda dari fungsi Lambda tradisional. Memahami konsep-konsep ini sangat penting untuk menerapkan dan mengelola Lambda Event Functions Anda secara efektif pada infrastruktur EC2.

Penyedia kapasitas membentuk dasar dari Lambda Managed Instances. Penyedia kapasitas menentukan infrastruktur komputasi tempat fungsi Anda dijalankan, termasuk konfigurasi VPC,

persyaratan instans, dan kebijakan penskalaan. Penyedia kapasitas juga berfungsi sebagai batas keamanan untuk fungsi Anda, yang berarti semua fungsi yang ditugaskan ke penyedia kapasitas yang sama harus saling dipercaya.

Perilaku penskalaan berbeda secara signifikan dari fungsi Lambda tradisional. Alih-alih melakukan penskalaan sesuai permintaan saat pemanggilan tiba, Instans Terkelola menskalakan secara asinkron berdasarkan pemanfaatan sumber daya CPU. Pendekatan ini menghilangkan start dingin tetapi membutuhkan perencanaan untuk pertumbuhan lalu lintas. Jika lalu lintas Anda lebih dari dua kali lipat dalam waktu 5 menit, Anda mungkin mengalami throttle karena Lambda meningkatkan kapasitas untuk memenuhi permintaan.

Keamanan dan izin membutuhkan pertimbangan yang cermat. Anda memerlukan izin peran operator untuk mengizinkan Lambda mengelola sumber daya EC2 di penyedia kapasitas Anda. Selain itu, pengguna membutuhkan `lambda:PassCapacityProvider` izin untuk menetapkan fungsi ke penyedia kapasitas, bertindak sebagai gerbang keamanan untuk mengontrol fungsi mana yang dapat berjalan pada infrastruktur tertentu.

Multi-concurrent eksekusi adalah karakteristik utama dari Managed Instances. Setiap lingkungan eksekusi dapat menangani beberapa pemanggilan secara bersamaan, memaksimalkan pemanfaatan sumber daya untuk IO-heavy aplikasi. Ini berbeda dari Lambda tradisional di mana setiap lingkungan memproses satu permintaan pada satu waktu. Model eksekusi ini membutuhkan perhatian pada keamanan utas, manajemen status, dan isolasi konteks tergantung pada runtime Anda.

Untuk detail tentang menerbitkan versi fungsi ke penyedia kapasitas Anda, lihat the section called “Penerbitan versi”.

Bagian berikut memberikan informasi terperinci tentang setiap konsep inti.

Penyedia kapasitas

Penyedia kapasitas adalah dasar untuk menjalankan Lambda Managed Instances. Ini bertindak sebagai batas keamanan untuk Lambda Event Functions Anda dan mendefinisikan sumber daya komputasi yang disediakan dan dikelola Lambda atas nama Anda.

Saat Anda membuat penyedia kapasitas, Anda menentukan:

- Konfigurasi VPC - Subnet dan grup keamanan tempat instance berjalan
- Izin - Peran IAM untuk Lambda untuk mengelola sumber daya EC2
- Persyaratan instans (opsional) - Arsitektur dan jenis contoh pilihan
- Konfigurasi penskalaan (opsional) - Bagaimana Lambda menskalakan instans Anda

Memahami penyedia kapasitas sebagai batas keamanan

Penyedia kapasitas berfungsi sebagai batas keamanan untuk fungsi Lambda dalam VPC Anda, menggantikan Firecracker-based isolasi. Fungsi dijalankan dalam wadah dalam instance, tetapi wadah tidak memberikan isolasi keamanan yang kuat antar fungsi, tidak seperti MicroVM Firecracker.

Konsep keamanan utama:

- Penyedia Kapasitas: Batas keamanan yang menentukan tingkat kepercayaan untuk fungsi Lambda
- Isolasi kontainer: Kontainer BUKAN penyedia keamanan - jangan mengandalkan mereka untuk keamanan antara beban kerja yang tidak tepercaya
- Pemisahan Kepercayaan: Pisahkan beban kerja yang tidak saling dipercaya dengan menggunakan penyedia kapasitas yang berbeda

Membuat Penyedia Kapasitas

Anda dapat membuat penyedia kapasitas menggunakan AWS CLI, AWS Konsol Manajemen, atau AWS SDK.

Menggunakan AWS CLI:

```
aws lambda create-capacity-provider \
  --capacity-provider-name my-capacity-provider \
  --vpc-config
 SubnetIds=subnet-12345,subnet-67890,subnet-11111,SecurityGroupIds=sg-12345 \
  --permissions-config CapacityProviderOperatorRoleArn=arn:aws:iam::123456789012:role/
MyOperatorRole \
  --instance-requirements Architectures=x86_64 \
  --capacity-provider-scaling-config ScalingMode=Auto
```

Parameter yang diperlukan

CapacityProviderName

- Nama unik untuk penyedia kapasitas Anda
- Harus unik di dalam diri Anda AWS akun

VpcConfig

- SubnetIds (wajib): Setidaknya satu subnet, maksimal 16. Gunakan subnet di beberapa Zona Ketersediaan untuk ketahanan
- SecurityGroupIds (opsional): Grup keamanan untuk instans Anda. Default ke grup keamanan default VPC jika tidak ditentukan

PermissionsConfig

- CapacityProviderOperatorRoleArn (wajib): Peran IAM yang memungkinkan Lambda mengelola sumber daya EC2 di penyedia kapasitas Anda

Parameter opsional

InstanceRequirements

Tentukan arsitektur dan jenis contoh untuk penyedia kapasitas Anda:

- Arsitektur: Pilih `x86_64` atau `arm64`. Defaultnya adalah `x86_64`
- AllowedInstanceTypes: Tentukan jenis instans yang diizinkan atau seluruh keluarga instance. Anda dapat menggunakan wildcard (`*`) untuk mencocokkan ukuran instance dalam keluarga tipe instance tertentu. Wildcard hanya berlaku untuk bagian ukuran (setelah titik). Contoh:
  - `m5.8xlarge` — Cocokkan dengan jenis instance yang tepat
  - `c6i.*` — Cocokkan dengan semua ukuran di `C6i` keluarga (`c6i.xlarge`, `c6i.2xlarge`, `c6i.4xlarge`, dan seterusnya)

  Karakter pengganti pada keluarga instans atau generasi tidak didukung. Sebagai contoh, `c5*.*`, `r*`, dan `*3*` bukan pola yang valid untuk Instans Terkelola Lambda.
- ExcludedInstanceTypes: Tentukan jenis instance untuk dikecualikan, menggunakan format yang sama seperti AllowedInstanceTypes. Anda hanya dapat menentukan satu AllowedInstanceTypes atau ExcludedInstanceTypes.

Secara default, Lambda memilih jenis instans optimal untuk beban kerja Anda. Sebaiknya biarkan Instans Terkelola Lambda memilih jenis instans untuk Anda, karena membatasi jumlah jenis instans yang mungkin dapat mengakibatkan ketersediaan lebih rendah.

CapacityProviderScalingConfig

Konfigurasikan cara Lambda menskalakan instans Anda:

- ScalingMode: Setel ke `Auto` untuk penskalaan otomatis atau `Manual` untuk kontrol manual. Defaultnya adalah `Auto`
- MaxVCpuCount: Jumlah maksimum vCPU untuk penyedia kapasitas. Defaultnya adalah 400.
- ScalingPolicies: Tentukan kebijakan penskalaan pelacakan target untuk pemanfaatan CPU dan memori

KmsKeyArn

Tentukan a AWS KMS kunci untuk enkripsi EBS. Defaultnya AWS kunci terkelola jika tidak ditentukan.

Tanda

Tambahkan tag untuk mengatur dan mengelola penyedia kapasitas Anda.

PropagateTags

Konfigurasikan propagasi tag untuk secara otomatis menerapkan tag ke sumber daya terkelola (instans Amazon EC2, volume Amazon EBS, dan ENI) yang diluncurkan oleh penyedia kapasitas. Gunakan propagasi tag untuk menegakkan alokasi biaya, SCP, dan persyaratan kepatuhan. Untuk informasi lebih lanjut, lihat Propagasi tag.

Mengelola penyedia kapasitas

Memperbarui penyedia kapasitas

Anda dapat memperbarui properti tertentu dari penyedia kapasitas menggunakan `UpdateCapacityProvider` API.

```
aws lambda update-capacity-provider \
  --capacity-provider-name my-capacity-provider \
  --capacity-provider-scaling-config ScalingMode=Auto
```

Menghapus penyedia kapasitas

Anda dapat menghapus penyedia kapasitas saat tidak lagi diperlukan menggunakan `DeleteCapacityProvider` API.

```
aws lambda delete-capacity-provider \
  --capacity-provider-name my-capacity-provider
```

Catatan: Anda tidak dapat menghapus penyedia kapasitas yang memiliki versi fungsi yang dilampirkan padanya.

Melihat detail penyedia kapasitas

Mengambil informasi tentang penyedia kapasitas menggunakan `GetCapacityProvider` API.

```
aws lambda get-capacity-provider \
  --capacity-provider-name my-capacity-provider
```

Negara penyedia kapasitas

Penyedia kapasitas dapat berada di salah satu negara berikut:

- Tertunda: Penyedia kapasitas sedang dibuat
- Aktif: Penyedia kapasitas siap digunakan
- Gagal: Pembuatan penyedia kapasitas gagal
- Menghapus: Penyedia kapasitas sedang dihapus

Kuota

- Penyedia kapasitas maksimum per akun: 1.000
- Versi fungsi maksimum per penyedia kapasitas: 100 (tidak dapat ditingkatkan)

Praktik Terbaik

1. Pisahkan berdasarkan tingkat kepercayaan: Buat penyedia kapasitas yang berbeda untuk beban kerja dengan persyaratan keamanan yang berbeda
2. Gunakan nama deskriptif: Sebutkan penyedia kapasitas untuk menunjukkan dengan jelas tujuan penggunaan dan tingkat kepercayaan mereka (misalnya, `production-trusted`, `dev-sandbox`).
3. Gunakan beberapa Zona Ketersediaan: Tentukan subnet di beberapa AZ untuk ketersediaan tinggi
4. Biarkan Lambda memilih jenis instance: Kecuali Anda memiliki persyaratan perangkat keras tertentu, izinkan Lambda memilih jenis instans yang optimal untuk ketersediaan
5. Memantau penggunaan: Gunakan AWS CloudTrail untuk memantau penugasan penyedia kapasitas dan pola akses

Langkah selanjutnya

- Pelajari tentang penskalaan Instans Terkelola Lambda
- Memahami keamanan dan izin untuk Instans Terkelola Lambda
- Konfigurasikan Konektivitas VPC untuk penyedia kapasitas Anda
- Tinjau panduan khusus runtime untuk Java, Node.js, dan Python

Perbanyakan tag

Dengan propagasi tag, Anda dapat menentukan satu set tag pada konfigurasi penyedia kapasitas, dan Lambda secara otomatis menerapkan tag tersebut ke semua sumber daya terkelola yang dibuatnya, termasuk instans Amazon EC2, volume Amazon EBS, dan ENI. Ini memastikan penandaan yang konsisten untuk alokasi biaya, kebijakan kontrol layanan (SCP), dan persyaratan kepatuhan tanpa memerlukan intervensi manual atau otomatisasi khusus.

Mengkonfigurasi propagasi tag

Tentukan `PropagateTags` setelan saat membuat atau memperbarui penyedia kapasitas menggunakan `UpdateCapacityProvider` API `CreateCapacityProvider` atau.

PropagateTags parameter:

- Mode (wajib) - Mode propagasi tag:
  - `Explicit`— Menyebarkan tag yang ditentukan `ExplicitTags` ke sumber daya yang dikelola.
  - `None`— Nonaktifkan propagasi tag. Tidak ada tag kustom yang diterapkan ke sumber daya terkelola kecuali tag sistem yang diterapkan secara default.
- ExplicitTags(diperlukan saat Mode`Explicit`) — Peta pasangan nilai kunci untuk diterapkan ke sumber daya yang dikelola. Anda dapat menentukan hingga 40 tag.

> **Important**
>
> Propagasi tag hanya berlaku untuk sumber daya terkelola baru yang disediakan setelah konfigurasi diterapkan. Sumber daya yang ada tidak ditandai secara surut.

Contoh

Buat penyedia kapasitas dengan propagasi tag (AWS CLI):

```
aws lambda create-capacity-provider \
  --capacity-provider-name my-capacity-provider \
  --vpc-config SubnetIds=subnet-12345,subnet-67890,SecurityGroupIds=sg-12345 \
  --permissions-config CapacityProviderOperatorRoleArn=arn:aws:iam::123456789012:role/
MyOperatorRole \
  --propagate-tags '{"Mode": "Explicit", "ExplicitTags": {"CostCenter": "12345",
 "Environment": "Production"}}'
```

Perbarui penyedia kapasitas yang ada untuk mengaktifkan propagasi tag:

```
aws lambda update-capacity-provider \
  --capacity-provider-name my-capacity-provider \
  --propagate-tags '{"Mode": "Explicit", "ExplicitTags": {"CostCenter": "12345",
 "Environment": "Production"}}'
```

Nonaktifkan propagasi tag:

```
aws lambda update-capacity-provider \
  --capacity-provider-name my-capacity-provider \
  --propagate-tags '{"Mode": "None"}'
```

Perilaku propagasi tag

- Perubahan `PropagateTags` konfigurasi hanya memengaruhi sumber daya yang disediakan setelah pembaruan. Sumber daya yang diluncurkan sebelumnya mempertahankan tag aslinya.
- Tag propagasi tag terpisah dari `Tags` parameter pada penyedia kapasitas itu sendiri. Tag penyedia kapasitas mengidentifikasi sumber daya penyedia kapasitas; tag yang disebarkan diterapkan ke sumber daya terkelola yang diluncurkan penyedia kapasitas.
- Tag yang disebarkan dihitung terhadap batas tag AWS sumber daya pada sumber daya target.

Menskalakan Instans Terkelola Lambda

Instans Terkelola Lambda tidak berskala saat pemanggilan tiba dan tidak mendukung cold start. Sebaliknya, ia menskalakan secara asinkron menggunakan sinyal konsumsi sumber daya. Instans Terkelola saat ini diskalakan berdasarkan pemanfaatan sumber daya CPU dan saturasi multi-konkurensi.

Perbedaan utama:

- Lambda (default): Menskalakan ketika tidak ada lingkungan eksekusi bebas untuk menangani pemanggilan masuk (start dingin)
- Instans Terkelola Lambda: Men skalakan secara asinkron berdasarkan pemanfaatan sumber daya CPU dan saturasi multi-konkurensi lingkungan eksekusi

Jika lalu lintas Anda lebih dari dua kali lipat dalam waktu 5 menit, Anda mungkin melihat pembatasan saat Lambda meningkatkan instans dan lingkungan eksekusi untuk memenuhi permintaan.

Siklus hidup penskalaan

Instans Terkelola Lambda menggunakan arsitektur terdistribusi untuk mengelola penskalaan:

Komponen:

- Instans Ter kelola - Jalankan di akun Anda di subnet yang Anda berikan
- Router dan Scaler - Komponen Lambda Bersama yang merutekan pemanggilan dan mengelola penskalaan
- Agen Lambda - Berjalan pada setiap Instans Terkelola untuk mengelola siklus hidup lingkungan eksekusi dan memantau konsumsi sumber daya

Cara kerjanya:

1. Saat Anda menerbitkan versi fungsi dengan penyedia kapasitas, Lambda meluncurkan Instans Terkelola di akun Anda. Ini meluncurkan tiga secara default untuk ketahanan AZ dan memulai tiga lingkungan eksekusi sebelum menandai versi fungsi Anda AKTIF.
2. Setiap Instans Terkelola dapat menjalankan lingkungan eksekusi untuk beberapa fungsi yang dipetakan ke penyedia kapasitas yang sama.
3. Saat lalu lintas mengalir ke aplikasi Anda, lingkungan eksekusi menghabiskan sumber daya. Agen Lambda memberi tahu Scaler, yang memutuskan apakah akan menskalakan lingkungan eksekusi baru atau Instans Terkelola.
4. Jika Router mencoba mengirim pemanggilan ke lingkungan eksekusi dengan konsumsi sumber daya tinggi, Agen Lambda pada instance tersebut akan memberitahukannya untuk mencoba lagi pada yang lain.
5. Saat lalu lintas berkurang, Agen Lambda memberi tahu Scaler, yang membuat keputusan untuk mengurangi lingkungan eksekusi dan skala dalam Instans Terkelola.

Menyesuaikan perilaku penskalaan

Anda dapat menyesuaikan perilaku penskalaan Instans Terkelola melalui lima kontrol:

Kontrol tingkat fungsi

1. Memori fungsi dan vCPU

Pilih ukuran memori dan alokasi vCPU untuk fungsi Anda. Ukuran fungsi terkecil yang didukung adalah 2GB dan 1vCPU.

Pertimbangan:

- Pilih pengaturan memori dan vCPU yang mendukung eksekusi multi-bersamaan dari fungsi Anda
- Anda tidak dapat mengonfigurasi fungsi dengan kurang dari 1 vCPU karena fungsi yang berjalan pada Instans Terkelola harus mendukung beban kerja multi-bersamaan
- Anda tidak dapat memilih kurang dari 2GB karena ini cocok dengan rasio memori 2 banding 1 terhadap vCPU dari instance c, yang memiliki rasio terendah
- Untuk aplikasi Python, Anda mungkin perlu memilih rasio memori yang lebih tinggi terhadap vCPU, seperti 4 banding 1 atau 8 banding 1, karena cara Python menangani multi-konkurensi
- Jika Anda menjalankan CPU-intensive operasi atau melakukan sedikit IO, Anda harus memilih lebih dari satu vCPU

2. Konkurensi maksimum

Tetapkan konkurensi maksimum per lingkungan eksekusi.

Perilaku default: Lambda memilih default masuk akal yang menyeimbangkan konsumsi sumber daya dan throughput yang berfungsi untuk berbagai aplikasi.

Pedoman penyesuaian:

- Tingkatkan konkurensi: Jika pemanggilan fungsi Anda menggunakan CPU yang sangat sedikit, Anda dapat meningkatkan konkurensi maksimum hingga maksimum 64 per vCPU
- Kurangi konkurensi: Jika aplikasi Anda mengkonsumsi sejumlah besar memori dan CPU yang sangat sedikit, Anda dapat mengurangi konkurensi maksimum Anda

Penting: Karena Instans Terkelola Lambda dimaksudkan untuk aplikasi multi-serentak, lingkungan eksekusi dengan konkurensi yang sangat rendah mungkin mengalami pembatasan saat penskalaan.

Ketika pemanggilan tiba di lingkungan eksekusi yang telah mencapai batas konkurensinya, Lambda merutekan pemanggilan tersebut ke tempat lain dan meningkatkan lingkungan eksekusi baru untuk menangani beban. Untuk mengidentifikasi batasan sumber daya mana yang menyebabkan throttle, pantau metrik alasan throttle (`ConcurrencyThrottles`,, `CPUThrottlesMemoryThrottles`, dan`DiskThrottles`) yang dijelaskan di. Jenis metrik untuk Fungsi Peristiwa Lambda

3. Lingkungan eksekusi per fungsi

Tetapkan jumlah minimum dan maksimum lingkungan eksekusi untuk fungsi Anda.

Perilaku default: Minimum default adalah 3 lingkungan eksekusi di seluruh Zona Ketersediaan, tanpa maksimum default. Anda dapat mengganti kedua nilai setelah fungsi dibuat.

Pedoman penyesuaian:

- Tetapkan minimum: Menye diakan kapasitas untuk lalu lintas dasar dan kurangi throttle selama semburan mendadak. Nilai di bawah 3 mengurangi redundansi Zona Ketersediaan.
- Tetapkan maksimum: Bat asi jumlah lingkungan eksekusi untuk mengontrol skala keluar dan mencegah masalah tetangga yang bising saat beberapa fungsi berbagi Penyedia Kapasitas.
- Nonaktifkan fungsi: Setel minimum dan maksimum ke 0 untuk menonaktifkan fungsi tanpa menghapusnya.

Contoh:

```
aws lambda put-function-scaling-config \
  --function-name my-lmi-function \
  --qualifier '$LATEST.PUBLISHED' \
  --function-scaling-config MinExecutionEnvironments=5,MaxExecutionEnvironments=20 \
  --region us-east-1
```

Catatan penting:

- Lingkup kualifikasi: Konfigurasi ini berlaku pada tingkat fungsi untuk setiap ARN yang memenuhi syarat. Saat diaktifkan`$LATEST.PUBLISHED`, konfigurasi menyebar ke `$LATEST.PUBLISHED` versi mendatang. Saat diatur pada versi tertentu, versi yang baru diterbitkan kembali ke nilai default.
- Konfigurasi berpasangan: Anda harus menyetel nilai minimum dan maksimum bersama-sama. Setiap pengaturan yang tidak ditentukan kembali ke nilai defaultnya. Nilai yang valid untuk

keduanya `MinExecutionEnvironments` dan ber `MaxExecutionEnvironments` kisar dari 0 hingga 15000. Minimal 0 hanya berlaku jika maksimum juga 0.

- Implikasi biaya: Pen onaktifan fungsi berlaku pada tingkat versi fungsi. Lambda mengakhiri instance EC2 yang mendasarinya setelah tidak memiliki lingkungan eksekusi aktif, dan biaya instans berlanjut hingga penghentian selesai (biasanya dalam beberapa menit).

Kontrol tingkat penyedia kapasitas

4. Target pemanfaatan sumber daya

Pilih target Anda sendiri untuk konsumsi pemanfaatan CPU.

Perilaku default: Lambda mempertahankan headroom yang cukup agar lalu lintas Anda berlipat ganda dalam waktu 5 menit tanpa pembatasan.

Opsi pengoptimalan:

- Jika beban kerja Anda sangat stabil atau jika aplikasi Anda tidak sensitif terhadap throttle, Anda dapat menetapkan target ke tingkat tinggi untuk mencapai pemanfaatan yang lebih tinggi dan biaya yang lebih rendah
- Jika Anda ingin mempertahankan ruang kepala untuk semburan lalu lintas, Anda dapat menetapkan target sumber daya ke tingkat rendah, yang membutuhkan lebih banyak kapasitas

5. Pemilihan tipe instans

Tetapkan jenis instans yang diizinkan atau dikecualikan.

Perilaku default: Lambda memilih jenis instans terbaik untuk beban kerja Anda. Dianjurkan untuk membiarkan Instans Terkelola Lambda memilih jenis instans, karena membatasi jumlah jenis instans yang mungkin dapat mengakibatkan ketersediaan yang lebih rendah.

Konfigurasi khusus:

- Persyaratan perangkat keras khusus: Tet apkan jenis instans yang diizinkan ke daftar instance yang kompatibel. Misalnya, jika Anda memiliki aplikasi yang membutuhkan bandwidth jaringan tinggi, Anda dapat memilih beberapa jenis n instance
- Pengoptimalan biaya: Untuk lingkungan pengujian atau pengembangan, Anda dapat memilih jenis instans yang lebih kecil, seperti jenis instans m7a.large

Penskalaan terjadwal

Gunakan Amazon EventBridge Scheduler untuk menyesuaikan lingkungan eksekusi minimum dan maksimum fungsi Anda pada jadwal berulang atau satu kali. Ini berguna untuk pola lalu lintas yang dapat diprediksi, seperti meningkatkan skala sebelum jam sibuk dan menurunkan skala selama jam di luar jam sibuk.

Konfigurasi penjadwal:

- Buat peran eksekusi EventBridge Scheduler atau gunakan peran yang ada yang memberikan izin untuk memanggil `lambda:PutFunctionScalingConfig` fungsi target Anda.
- Buat jadwal menggunakan ekspresi cron atau rate, menargetkan `PutFunctionScalingConfig` API sebagai target universal. Tentukan `MaxExecutionEnvironments` nilai baru `MinExecutionEnvironments` dan dalam muatan Input.

Contoh 1: Skala untuk menangani lalu lintas puncak yang direncanakan

Buat dua jadwal untuk meningkatkan skala sebelum jam sibuk dan kurangi setelahnya. Setiap jadwal menarget `PutFunctionScalingConfig` kan API dengan pembaruan `MinExecutionEnvironments` dan `MaxExecutionEnvironments` nilai.

Tingkatkan skala pada pukul 8:00 UTC (min=100, maks=1000):

```
aws scheduler create-schedule \
  --name "ScaleUpLambdaManagedInstances" \
  --schedule-expression "cron(0 8 * * ? *)" \
  --flexible-time-window '{"Mode": "OFF"}' \
  --target '{
    "Arn": "arn:aws:scheduler:::aws-sdk:lambda:PutFunctionScalingConfig",
    "RoleArn": "arn:aws:iam::<account-id>:role/eventbridge-scheduler-role",
    "Input": "{\"FunctionName\": \"my-lmi-function\", \"Qualifier\":
 \"$LATEST.PUBLISHED\", \"FunctionScalingConfig\": {\"MinExecutionEnvironments\": 100,
 \"MaxExecutionEnvironments\": 1000}}"
  }'
```

Menurunkan skala pada pukul 18:00 UTC (min=5, maks=20):

```
aws scheduler create-schedule \
  --name "ScaleDownLambdaManagedInstances" \
  --schedule-expression "cron(0 18 * * ? *)" \
  --flexible-time-window '{"Mode": "OFF"}' \
```

```
  --target '{
    "Arn": "arn:aws:scheduler:::aws-sdk:lambda:PutFunctionScalingConfig",
    "RoleArn": "arn:aws:iam::<account-id>:role/eventbridge-scheduler-role",
    "Input": "{\"FunctionName\": \"my-lmi-function\", \"Qualifier\":
 \"$LATEST.PUBLISHED\", \"FunctionScalingConfig\": {\"MinExecutionEnvironments\": 5,
 \"MaxExecutionEnvironments\": 20}}"
  }'
```

Contoh 2: Nonaktifkan selama jam di luar jam sibuk dan aktifkan kembali

Menyetel keduanya `MinExecutionEnvironments` dan `MaxExecutionEnvironments` ke 0 menonaktifkan versi fungsi tanpa menghapusnya. Fungsi yang dinonaktifkan tidak secara otomatis meningkatkan kembali dengan lalu lintas. Anda harus secara eksplisit mengaktifkannya kembali dengan menetapkan nilai bukan nol melalui tindakan terjadwal lainnya.

Nonaktifkan pada 10:00 PM UTC (min=0, max=0):

```
aws scheduler create-schedule \
  --name "DeactivateLambdaManagedInstances" \
  --schedule-expression "cron(0 22 * * ? *)" \
  --flexible-time-window '{"Mode": "OFF"}' \
  --target '{
    "Arn": "arn:aws:scheduler:::aws-sdk:lambda:PutFunctionScalingConfig",
    "RoleArn": "arn:aws:iam::<account-id>:role/eventbridge-scheduler-role",
    "Input": "{\"FunctionName\": \"my-lmi-function\", \"Qualifier\":
 \"$LATEST.PUBLISHED\", \"FunctionScalingConfig\": {\"MinExecutionEnvironments\": 0,
 \"MaxExecutionEnvironments\": 0}}"
  }'
```

Aktifkan kembali pada pukul 7:00 UTC (min=10, maks=20):

```
aws scheduler create-schedule \
  --name "ReactivateLambdaManagedInstances" \
  --schedule-expression "cron(0 7 * * ? *)" \
  --flexible-time-window '{"Mode": "OFF"}' \
  --target '{
    "Arn": "arn:aws:scheduler:::aws-sdk:lambda:PutFunctionScalingConfig",
    "RoleArn": "arn:aws:iam::<account-id>:role/eventbridge-scheduler-role",
    "Input": "{\"FunctionName\": \"my-lmi-function\", \"Qualifier\":
 \"$LATEST.PUBLISHED\", \"FunctionScalingConfig\": {\"MinExecutionEnvironments\": 10,
 \"MaxExecutionEnvironments\": 20}}"
  }'
```

Pedoman penyesuaian:

- Untuk beban kerja dengan puncak yang dapat diprediksi, buat beberapa jadwal agar sesuai dengan pola lalu lintas Anda: satu untuk meningkatkan fungsi Anda sebelum jam sibuk, dan satu lagi untuk menurunkan skala setelah jam sibuk. Setiap jadwal mengikuti pola yang sama dengan pembaruan `MinExecutionEnvironments` dan `MaxExecutionEnvironments` nilai.
- Penskalaan terjadwal menyesuaikan lantai dan plafon lingkungan eksekusi yang disediakan, tetapi penskalaan aktual antara min dan max masih merespons pemanfaatan CPU dan saturasi konkurensi.
- Jika lalu lintas Anda lebih dari dua kali lipat dalam waktu 5 menit setelah peningkatan skala yang dijadwalkan, Anda mungkin masih mengalami pembatasan saat kapasitas disediakan.
- Saat menskalakan ke nol untuk menonaktifkan fungsi, ingatlah bahwa reaktivasi memerlukan `PutFunctionScalingConfig` panggilan eksplisit dengan nilai bukan nol.

Langkah selanjutnya

- Pelajari tentang penyedia kapasitas untuk Instans Terkelola Lambda
- Tinjau panduan khusus runtime untuk menangani multi-konkurensi
- Konfigurasikan konektivitas VPC untuk penyedia kapasitas Anda
- Memantau metrik penskalaan untuk mengoptimalkan perilaku penskalaan

Keamanan dan izin

Instans Terkelola Lambda menggunakan penyedia kapasitas sebagai batas kepercayaan. Fungsi Peristiwa Lambda dijalankan dalam kontainer dalam instance ini, tetapi kontainer tidak menyediakan isolasi keamanan antar beban kerja. Semua fungsi yang ditugaskan ke penyedia kapasitas yang sama harus saling dipercaya.

Konsep Keamanan Utama

- Penyedia Kapasitas: Batas keamanan yang menentukan tingkat kepercayaan untuk fungsi Lambda
- Isolasi Kontainer: Kontainer bukan batas keamanan - jangan mengandalkannya untuk keamanan di antara beban kerja yang tidak tepercaya
- Pemisahan Kepercayaan: Pisahkan beban kerja yang tidak saling dipercaya dengan menggunakan penyedia kapasitas yang berbeda

Izin yang Diperlukan

PassCapacityProvider Aksi

Pengguna memerlukan `lambda:PassCapacityProvider` izin untuk menetapkan fungsi ke penyedia kapasitas. Izin ini bertindak sebagai gerbang keamanan, memastikan hanya pengguna yang berwenang yang dapat menempatkan fungsi di penyedia kapasitas tertentu.

Administrator akun mengontrol fungsi mana yang dapat menggunakan penyedia kapasitas tertentu melalui tindakan `lambda:PassCapacityProvider` IAM. Tindakan ini diperlukan ketika:

- Membuat fungsi yang menggunakan Instans Terkelola Lambda
- Memperbarui konfigurasi fungsi untuk menggunakan penyedia kapasitas
- Menerapkan fungsi dengan menggunakan infrastruktur sebagai kode

Contoh Kebijakan IAM

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "lambda:PassCapacityProvider",
      "Resource": "arn:aws:lambda:*:*:capacity-provider:trusted-workloads-*"
    }
  ]
}
```

Service-Linked Peran

AWS Lambda menggunakan peran `AWSServiceRoleForLambda` terkait layanan untuk mengelola sumber daya ec2 Instans Terkelola Lambda di penyedia kapasitas Anda.

Praktik Terbaik

1. Pisahkan berdasarkan Tingkat Kepercayaan: Buat penyedia kapasitas yang berbeda untuk beban kerja dengan persyaratan keamanan yang berbeda
2. Gunakan Nama Deskriptif: Beri nama penyedia kapasitas untuk menunjukkan dengan jelas tujuan penggunaan dan tingkat kepercayaan mereka (misalnya,`production-trusted`,`dev-sandbox`)

3. Terapkan Hak Istimewa Paling Sedikit: Berikan `PassCapacityProvider` izin hanya untuk penyedia kapasitas yang diperlukan
4. Penggunaan Monitor: Gunakan AWS CloudTrail untuk memantau penetapan penyedia kapasitas dan pola akses

Langkah selanjutnya

- Pelajari tentang penyedia kapasitas untuk Instans Terkelola Lambda
- Memahami penskalaan untuk Instans Terkelola Lambda
- Konfigurasikan konektivitas VPC untuk penyedia kapasitas Anda
- Tinjau panduan khusus runtime untuk Java, Node.js, dan Python

Peran operator Lambda untuk Instans Terkelola Lambda

Saat Anda menggunakan Instans Terkelola Lambda, Lambda memerlukan izin untuk mengelola kapasitas komputasi di akun Anda. Peran operator memberikan izin ini melalui kebijakan IAM yang memungkinkan Lambda mengelola instans EC2 di penyedia kapasitas.

Lambda mengasumsikan peran operator saat melakukan operasi manajemen ini, mirip dengan cara Lambda mengasumsikan peran eksekusi saat Fungsi Peristiwa Lambda Anda berjalan.

Membuat peran operator

Anda dapat membuat peran operator di konsol IAM atau dengan AWS CLI. Peran tersebut harus mencakup:

- Kebijakan izin - Memberikan izin untuk mengelola penyedia kapasitas dan sumber daya terkait
- Kebijakan kepercayaan - Memungkinkan layanan Lambda (`lambda.amazonaws.com`) untuk mengambil peran

Kebijakan izin

Peran operator memerlukan izin untuk mengelola penyedia kapasitas dan sumber daya komputasi yang mendasarinya. Minimal, peran memerlukan izin dalam kebijakan AWSLambdaManagedEC2ResourceOperatorterkelola, saat ini:

```
{
  "Version": "2012-10-17",
```

```
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:RunInstances",
        "ec2:CreateTags",
        "ec2:AttachNetworkInterface"
      ],
      "Resource": [
        "arn:aws:ec2:*:*:instance/*",
        "arn:aws:ec2:*:*:network-interface/*",
        "arn:aws:ec2:*:*:volume/*"
      ],
      "Condition": {
        "StringEquals": {
          "ec2:ManagedResourceOperator": "scaler.lambda.amazonaws.com"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
        "ec2:DescribeAvailabilityZones",
        "ec2:DescribeCapacityReservations",
        "ec2:DescribeInstances",
        "ec2:DescribeInstanceStatus",
        "ec2:DescribeInstanceTypeOfferings",
        "ec2:DescribeInstanceTypes",
        "ec2:DescribeSecurityGroups",
        "ec2:DescribeSubnets",
        "ec2:DescribeVpcEncryptionControls"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "ec2:RunInstances",
        "ec2:CreateNetworkInterface"
      ],
      "Resource": [
        "arn:aws:ec2:*:*:subnet/*",
        "arn:aws:ec2:*:*:security-group/*"
      ]
```

```
    },
    {
      "Effect": "Allow",
      "Action": [
        "ec2:RunInstances"
      ],
      "Resource": [
        "arn:aws:ec2:*:*:image/*"
      ],
      "Condition": {
        "StringEquals": {
          "ec2:Owner": "amazon"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup"
      ],
      "Resource": "arn:aws:logs:*:*:log-group:*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": "arn:aws:logs:*:*:log-group:*:log-stream:*"
    }
  ]
}
```

Kebijakan kepercayaan

Kebijakan kepercayaan memungkinkan Lambda untuk mengambil peran operator:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "lambda.amazonaws.com"
```

```
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Service-Linked Peran untuk Instans Terkelola Lambda

Untuk mengelola siklus hidup Instans Terkelola Lambda secara bertanggung jawab, Lambda memerlukan akses terus-menerus untuk menghentikan instans terkelola di akun Anda. Lambda menggunakan AWS Identity and Access Management (IAM) and Access Management (SLR) service-linked role (SLR) untuk melakukan operasi ini.

Pembuatan otomatis: Peran terkait layanan dibuat secara otomatis saat pertama kali Anda membuat penyedia kapasitas. Pengguna yang membuat penyedia kapasitas pertama harus memiliki `iam:CreateServiceLinkedRole` izin untuk kepala `lambda.amazonaws.com` sekolah.

Izin: Peran terkait layanan memberi Lambda izin berikut pada instans terkelola:

- `ec2:TerminateInstances`— Untuk mengakhiri instance di akhir siklus hidupnya
- `ec2:DescribeInstances`— Untuk menghitung instance terkelola

Penghapusan: Anda hanya dapat menghapus peran terkait layanan ini setelah menghapus semua penyedia kapasitas Instans Terkelola Lambda di akun Anda.

Untuk mengetahui informasi selengkapnya tentang peran terkait layanan, lihat the section called “Menggunakan peran tertaut layanan”.

Memahami lingkungan eksekusi Instans Terkelola Lambda

Instans Terkelola Lambda menyediakan model penerapan alternatif yang menjalankan kode fungsi Anda pada instans Amazon EC2 milik pelanggan sementara Lambda mengelola aspek operasional. Lingkungan eksekusi untuk Instans Terkelola memiliki beberapa perbedaan penting dari fungsi Lambda (default), terutama dalam cara menangani pemanggilan bersamaan dan mengelola siklus hidup kontainer.

Catatan: Untuk informasi tentang lingkungan eksekusi Lambda (default), lihat Memahami siklus hidup lingkungan eksekusi Lambda.

Siklus hidup lingkungan eksekusi

Siklus hidup lingkungan eksekusi fungsi Instans Terkelola Lambda berbeda dari Lambda (default) dalam beberapa cara utama:

Fase inisialisasi

Selama fase Init, Lambda melakukan langkah-langkah berikut:

- Inisialisasi dan daftarkan semua ekstensi
- Bootstrap titik masuk runtime. Runtime memunculkan jumlah pekerja runtime yang dikonfigurasi (implementasi tergantung pada runtime)
- Jalankan kode inisialisasi fungsi (kode di luar handler)
- Tunggu setidaknya satu pekerja runtime untuk memberi sinyal kesiapan dengan menelepon `/runtime/invocation/next`

Fase Init dianggap selesai ketika ekstensi telah diinisialisasi dan setidaknya satu runtime worker telah memanggil`/runtime/invocation/next`. Fungsi tersebut kemudian siap untuk memproses pemanggilan.

> **Note**
>
> Untuk fungsi Instans Terkelola Lambda, inisialisasi dapat memakan waktu hingga 15 menit. Waktu tunggu adalah maksimum 130 detik atau batas waktu fungsi yang dikonfigurasi (hingga 900 detik).

Fase invokasi

Fase Invoke untuk fungsi Instans Terkelola Lambda memiliki beberapa karakteristik unik:

Operasi terus menerus. Tidak seperti Lambda (default), lingkungan eksekusi tetap aktif terus menerus, memproses pemanggilan saat tiba tanpa membeku di antara pemanggilan.

Pemrosesan paralel. Beberapa pemanggilan dapat dijalankan secara bersamaan dalam lingkungan eksekusi yang sama, masing-masing ditangani oleh runtime worker yang berbeda.

Waktu tunggu independen. Batas waktu yang dikonfigurasi fungsi berlaku untuk setiap pemanggilan individu. Ketika waktu pemanggilan habis, Lambda menandai pemanggilan tertentu sebagai gagal

tetapi tidak mengganggu pemanggilan lain yang sedang berjalan atau menghentikan lingkungan eksekusi.

Penanganan tekanan balik. Jika semua pekerja runtime sibuk memproses pemanggilan, permintaan pemanggilan baru ditolak sampai pekerja tersedia.

Penanganan dan pemulihan kesalahan

Penanganan kesalahan di lingkungan eksekusi fungsi Instans Terkelola Lambda berbeda dari Lambda (default):

Kegagalan pekerja runtime. Jika proses runtime worker macet, lingkungan eksekusi terus beroperasi dengan pekerja sehat yang tersisa.

Ekstensi mogok. Jika proses ekstensi macet selama inisialisasi atau operasi, seluruh lingkungan eksekusi ditandai sebagai tidak sehat dan dihentikan. Lambda menciptakan lingkungan eksekusi baru untuk menggantikannya.

Tidak reset/repair. Tidak seperti Lambda (default), Instans Terkelola tidak mencoba mengatur ulang dan menginisialisasi ulang lingkungan eksekusi setelah kesalahan. Sebaliknya, wadah yang tidak sehat dihentikan dan diganti dengan yang baru.

Memanggil batas waktu

Ketika waktu pemanggilan individu habis, Lambda mengembalikan `Task timed out after <timeout> seconds` kesalahan dengan status kesalahan fungsi ke pemanggil. Namun, Instans Terkelola Lambda tidak secara paksa mengakhiri kode Anda—kode tersebut terus berjalan di lingkungan eksekusi. Sebagai pengembang fungsi, Anda bertanggung jawab untuk mendeteksi dan menangani batas waktu.

Objek konteks memperlihatkan sisa waktu untuk pemanggilan. Nilai nol atau negatif menunjukkan waktu pemanggilan telah habis. Pemanggilan bersamaan lainnya di lingkungan eksekusi terus diproses secara normal.

Perilaku coba lagi

Ketika waktu pemanggilan habis:

- Pemanggilan sinkron: Penelep on menerima kesalahan batas waktu dan bertanggung jawab untuk mencoba lagi.

- Pemanggilan asinkron: Percobaan ulang Lambda berdasarkan kebijakan coba ulang fungsi Anda (default: 2 percobaan ulang). Setelah semua percobaan ulang habis, acara dikirim ke antrean huruf mati yang dikonfigurasi atau tujuan gagal, jika ada.
- Pemetaan sumber peristiwa: Peril aku coba lagi tergantung pada konfigurasi sumber peristiwa (misalnya, ukuran batch, membagi kesalahan, upaya coba ulang maksimum). Batch dapat dicoba ulang atau dikirim ke tujuan saat gagal berdasarkan kebijakan coba ulang Anda.

Apa yang terjadi jika Anda tidak menangani batas waktu

Jika kode Anda tidak memeriksa sisa waktu dan menghentikan eksekusi:

- Pemanggilan sudah ditandai sebagai gagal. Lambda telah mengembalikan kesalahan batas waktu ke pemanggil—pekerjaan apa pun yang diselesaikan kode Anda setelah batas waktu secara efektif hilang dari sudut pandang pemanggil.
- Sumber daya tetap dikonsumsi. Kode Anda terus menempati slot pekerja runtime, mengurangi konkurensi yang tersedia untuk pemanggilan baru pada instance tersebut.
- Perilaku nondeterministik. Kode Anda tidak berhenti ketika batas waktu meledak—kode ini terus berjalan di latar belakang. Ini berarti efek samping masih dapat terjadi setelah Lambda telah memberi tahu penelepon bahwa pemanggilan gagal.

  Misalnya, handler Anda menulis catatan ke DynamoDB, kemudian batas waktu diaktifkan dan Lambda mengembalikan kesalahan batas waktu ke pemanggil, tetapi kode Anda masih berjalan dan melanjutkan untuk mengirim pemberitahuan SNS. Penelepon mencoba lagi pemanggilan, yang menulis catatan lagi dan mengirim pemberitahuan lagi. Anda sekarang memiliki data duplikat dan pemberitahuan duplikat—dan tidak ada cara mudah untuk mengetahui mana yang berasal dari pemanggilan “gagal” yang masih berjalan di latar belakang.

Menangani batas waktu dalam kode Anda

Gunakan objek konteks untuk memeriksa sisa waktu dan menghentikan pemrosesan sebelum batas waktu. Konfigurasikan buffer berdasarkan durasi yang diharapkan dari unit kerja Anda berikutnya —misalnya, jika setiap item membutuhkan waktu sekitar 500 ms untuk diproses, setel buffer ke setidaknya 500 ms plus margin.

Untuk contoh penanganan batas waktu khusus bahasa, lihat bagian konteks permintaan dari setiap halaman runtime:

- Node.js konteks permintaan

- Konteks permintaan Python
- Konteks permintaan Java
- Konteks perminta.NET

Lazy-load gambar kontainer dengan Seekable OCI (SOCI)

Untuk Lambda Event Functions yang digunakan sebagai image container pada Lambda Managed Instances, Lambda mengunduh dan membongkar gambar pada setiap instance sebelum memulai lingkungan eksekusi. Untuk gambar besar, tarikan ini adalah bagian terbesar dari waktu antara menerbitkan versi fungsi atau memperkecil skala dan lingkungan eksekusi menangani pemanggilan pertamanya. Dengan Seekable OCI (SOCI), Lambda dapat memulai lingkungan eksekusi sebelum seluruh gambar diunduh.

SOCI adalah sebuah teknologi open source dikembangkan oleh AWS. Ini menggunakan indeks yang memetakan setiap file dalam gambar ke lokasinya di lapisan gambar. Lambda menggunakan indeks untuk mengambil hanya file sesuai permintaan saat fungsi Anda membacanya, alih-alih mengunduh dan membongkar gambar lengkap sebelum memulai wadah. Gambar kontainer biasanya menggabungkan kumpulan pustaka dan dependensi yang luas, tidak semuanya dibaca selama startup atau pemanggilan tunggal. Lazy loading memanfaatkan ini dengan hanya mengambil apa yang sebenarnya dibaca fungsi Anda, mengurangi waktu hingga pemanggilan pertama dan membuat waktu startup jauh lebih tidak sensitif terhadap ukuran gambar total. Manfaatnya terbesar untuk gambar beberapa gigabyte di mana fungsi Anda hanya membaca sebagian konten per pemanggilan, seperti gambar yang mengemas kerangka kerja pembelajaran mesin.

Cara kerja SOCI pada Instans Terkelola Lambda

1. Anda membuat indeks SOCI untuk gambar Anda dan mendorongnya ke Amazon ECR di samping gambar. Hasilnya adalah indeks gambar Open Container Initiative (OCI) yang mereferensikan dua artefak: manifes gambar asli Anda dan indeks SOCI.
2. Anda membuat atau memperbarui fungsi Anda dengan URI gambar yang menunjuk ke indeks gambar. Lambda mendeteksi indeks SOCI dan memungkinkan pemuatan lambat untuk lingkungan eksekusi fungsi itu. Anda tidak perlu mengubah konfigurasi fungsi lainnya.
3. Lambda mengambil konten gambar sesuai permintaan. File yang dibaca kode inisialisasi Anda diambil selama inisialisasi. File yang dibaca handler Anda untuk pertama kalinya diambil selama pemanggilan itu. Lambda tidak mengunduh sisa gambar di latar belakang.

> **Note**
>
> Jika fungsi Anda membaca file untuk pertama kalinya selama pemanggilan daripada selama inisialisasi, Lambda mengambil file itu pada pemanggilan pertama, yang meningkatkan durasi pemanggilan itu. Agar durasi pemanggilan tetap konsisten, impor semua pustaka dan baca semua file yang dibutuhkan fungsi Anda dalam kode inisialisasi Anda, bukan di handler. Kami menyarankan untuk menguji fungsi Anda dengan dan tanpa SOCI untuk membandingkan waktu startup dan durasi pemanggilan untuk beban kerja Anda.

Buat indeks SOCI untuk gambar Anda

Contoh berikut berjalan di Amazon Linux 2023. Ini menginstal SOCI snapshotter, containerd, dan nerdctl, kemudian membuat indeks SOCI untuk gambar yang ada di Amazon ECR dan mendorong hasilnya. Ganti nilai placeholder dengan milik Anda sendiri. Untuk cara lain untuk membuat dan mendorong indeks SOCI, lihat Memulai dengan snapshotter SOCI di atas GitHub.

```
sudo yum install -y soci-snapshotter containerd nerdctl
sudo systemctl start soci-snapshotter
sudo systemctl restart containerd

ACCOUNT_ID="111122223333"
REGION="us-east-2"
REPOSITORY_NAME="my-function"
ORIGINAL_IMAGE_TAG="v1"
SOCI_IMAGE_TAG="v1-soci"
REGISTRY="$ACCOUNT_ID.dkr.ecr.$REGION.amazonaws.com"

aws ecr get-login-password --region $REGION | sudo nerdctl login -u AWS --password-
stdin $REGISTRY

sudo nerdctl pull $REGISTRY/$REPOSITORY_NAME:$ORIGINAL_IMAGE_TAG
sudo nerdctl image convert --soci $REGISTRY/$REPOSITORY_NAME:$ORIGINAL_IMAGE_TAG
 $REGISTRY/$REPOSITORY_NAME:$SOCI_IMAGE_TAG
sudo nerdctl push --platform linux/arm64 $REGISTRY/$REPOSITORY_NAME:$SOCI_IMAGE_TAG
```

> **Note**
>
> Contoh ini membangun sebuah `arm64` (AWS Gambar Graviton). Lambda hanya mendukung gambar wadah arsitektur tunggal, sehingga gambar Anda harus menargetkan arsitektur set

instruksi tunggal, dan `--platform` nilai harus sesuai dengan arsitektur fungsi Anda. Untuk informasi selengkapnya, lihat the section called “Set instruksi (ARM/x86)”.

Gunakan `$REGISTRY/$REPOSITORY_NAME:$SOCI_IMAGE_TAG` sebagai URI gambar saat Anda membuat atau memperbarui fungsi Anda. Kemudian publikasikan versi fungsi. Lambda menyelesaikan versi fungsi ke indeks gambar dan memungkinkan pemuatan lambat.

Persyaratan dan pembatasan

Versi indeks SOCI

Lambda hanya mendukung SOCI Index Manifest v2 (tipe media `application/vnd.amazon.soci.index.v2+json`). Lambda tidak mendukung indeks SOCI Index Manifest v1. Yang `nerdctl image convert --soci` perintah di bagian sebelumnya menghasilkan indeks v2.

Format indeks gambar

Indeks gambar harus berisi tepat dua entri: indeks SOCI dan satu manifes gambar OCI.

URI citra

Arahkan URI gambar fungsi Anda ke SOCI-indexed tag (misalnya, `my-function:v1-soci`), bukan pada tag gambar asli. Lambda menyelesaikan versi fungsi ke indeks gambar. Untuk mengubah gambar atau indeks, tekan yang baru SOCI-indexed tag dan perbarui URI gambar fungsi untuk itu.

Ketersediaan

SOCI hanya tersedia untuk fungsi image container yang berjalan pada Lambda Managed Instances.

LATEST.PUBLISHED Versi $ di Instans Terkelola Lambda

Fungsi Instans Terkelola Lambda mendukung alur kerja versi bernomor yang sama dengan Lambda (default). Jika Anda memilih untuk tidak mempertahankan versi bernomor, Lambda Managed Instances memperkenalkan jenis versi baru:`$LATEST.PUBLISHED`. Dengan jenis versi ini, Anda dapat membuat atau menerbitkan ulang versi yang diterbitkan dengan kode atau konfigurasi yang diperbarui. Anda tidak perlu mengelola versi bernomor.

Perbedaan utama dari $LATEST: Saat Anda memanggil fungsi Instans Terkelola Lambda menggunakan ARN yang tidak memenuhi syarat, Lambda secara implisit memanggil versi daripada versi $LATEST yang tidak dipublikasikan. `$LATEST.PUBLISHED`

Perintah AWS CLI berikut membuat atau mereproduksi versi. `$LATEST.PUBLISHED`

```
aws lambda publish-version --function-name my-function --publish-to LATEST_PUBLISHED
```

Anda akan melihat output berikut:

```
{
  "FunctionName": "my-function",
  "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-function:
$LATEST.PUBLISHED",
  "Version": "$LATEST.PUBLISHED",
  "Role": "arn:aws:iam::123456789012:role/lambda-role",
  "Handler": "function.handler",
  "Runtime": "nodejs24.x",
  ...
}
```

> **Note**
>
> Jika Anda menggunakan AWS CloudFormation atau konsol Lambda untuk membuat fungsi Instans Terkelola Lambda, Lambda akan membuat versi secara otomatis. `$LATEST.PUBLISHED`

Menyebarkan dan memperbarui fungsi Instans Terkelola Lambda

Saat Anda memperbarui fungsi Instans Terkelola Lambda, Lambda menyediakan lingkungan eksekusi untuk versi fungsi baru pada instans terkelola sebelum mengalihkan lalu lintas ke sana. Anda dapat mengontrol kecepatan dan keamanan penerapan dengan memilih strategi penerapan. Topik ini menjelaskan strategi penerapan yang tersedia untuk fungsi Instans Terkelola Lambda dan cara mendefinisikannya dengan infrastruktur sebagai kode (IaC).

> **Note**
>
> Sebelum Anda menerapkan fungsi, Anda harus memiliki penyedia kapasitas. Untuk informasi lebih lanjut, lihat penyedia kapasitas untuk Instans Terkelola Lambda.

Strategi penyebaran

Instans Terkelola Lambda mendukung strategi berikut untuk merilis versi fungsi baru. Pilih strategi yang sesuai dengan seberapa banyak kontrol yang ingin Anda pertahankan selama peluncuran. Pembaruan bergulir adalah opsi paling sederhana. Blue/green dan penerapan canary memberi Anda kontrol lebih besar atas bagaimana lalu lintas bergerak ke versi baru.

Pembaruan bergulir

Untuk melakukan pembaruan bergulir, Anda mempublikasikan perubahan ke `$LATEST.PUBLISHED` versi fungsi Anda. `$LATEST.PUBLISHED` adalah jenis versi khusus untuk Instans Terkelola Lambda yang Anda publikasikan ulang dengan kode atau konfigurasi baru alih-alih mempertahankan versi bernomor. Saat Anda memanggil fungsi Instans Terkelola Lambda dengan Nama Sumber Daya Amazon (ARN) yang tidak memenuhi syarat, Lambda memanggil `$LATEST.PUBLISHED` versi. Untuk informasi selengkapnya, lihat the section called “Penerbitan versi”.

Setiap kali Anda menerbitkan ulang `$LATEST.PUBLISHED`, Lambda secara bertahap menggantikan lingkungan eksekusi pada instans terkelola Anda dengan versi baru, tanpa mengharuskan Anda mengelola pergeseran lalu lintas secara langsung melalui alias fungsi. Ini adalah strategi paling sederhana.

Gunakan pembaruan bergulir saat Anda menginginkan pergeseran lalu lintas bertahap otomatis dari versi lama ke versi baru, dan Anda tidak memerlukan kontrol langsung atas persentase lalu lintas yang dikirim ke setiap versi untuk mengelola penerapan atau rollback.

Blue/green penerapan dengan alias tertimbang

Dalam sebuah blue/green penerapan, Anda menerbitkan versi baru bersama versi saat ini dan menggunakan alias dengan konfigurasi routing untuk mengirim persentase lalu lintas ke versi baru. Anda memanggil fungsi melalui ARN yang memenuhi syarat alias. Untuk memutar penerapan ke depan, Anda menambah bobot pada versi baru; untuk memutarnya kembali, Anda menggeser bobot kembali ke versi sebelumnya.

Contoh berikut menggunakan AWS CLI untuk mengirim 10 persen lalu lintas ke versi 2 sementara versi 1 terus melayani 90 persen sisanya.

```
aws lambda update-alias \
  --function-name my-managed-instance-function \
  --name production \
  --function-version 1 \
  --routing-config AdditionalVersionWeights={"2"=0.1}
```

Dalam perintah ini:

- `my-managed-instance-function` adalah nama fungsi Anda.
- `production` adalah nama alias yang Anda panggil.
- `--function-version 1` adalah versi yang terus menerima lalu lintas yang tersisa.
- `AdditionalVersionWeights={"2"=0.1}` mengirimkan 10 persen lalu lintas ke versi 2.

Untuk memutar kembali, hapus konfigurasi routing sehingga semua lalu lintas kembali ke versi stabil.

```
aws lambda update-alias \
  --function-name my-managed-instance-function \
  --name production \
  --function-version 1 \
  --routing-config '{}'
```

Dalam perintah ini, `--routing-config '{}'` menghapus konfigurasi routing sehingga semua lalu lintas kembali ke versi di `--function-version`, versi `1`.

Penyebaran Canary dengan AWS CodeDeploy

Untuk mengotomatiskan a blue/green pergeseran lalu lintas, gunakan AWS CodeDeploy. CodeDeploy mengalihkan lalu lintas dari versi saat ini ke versi baru sesuai dengan konfigurasi penerapan—misalnya, mengirim 10 persen lalu lintas ke versi baru, menunggu lima menit, dan kemudian memindahkan sisanya. Jika a CloudWatch alarm yang terkait dengan penyebaran memasuki keadaan alarm selama shift, CodeDeploy menggulung penerapan kembali secara otomatis.

Untuk informasi lebih lanjut tentang CodeDeploy konfigurasi penerapan untuk Lambda, lihat Konfigurasi penerapan pada platform komputasi Lambda di AWS CodeDeploy Panduan Pengguna.

Memilih antara $LATEST.PUBLISHED dan versi numerik dengan alias

Kedua model penerapan meluncurkan perubahan bertahap. Apakah Anda memperbarui `$LATEST.PUBLISHED` atau menggeser lalu lintas dengan alias tertimbang, Lambda Managed Instances menggantikan lingkungan eksekusi versi lama dengan versi baru secara bertahap, bukan sekaligus. Pendekatan ini membuat pemanfaatan instans tetap efisien selama peluncuran. Alih-alih menyediakan satu set lengkap lingkungan eksekusi kedua di muka, Lambda menggantinya secara bertahap: pada setiap langkah ia mengklaim kembali kapasitas dari versi lama sebelum memunculkan batch berikutnya dari versi baru, sehingga peluncuran menghindari mencapai dua kali lipat kapasitas dasar.

Bagaimana $LATEST.PUBLISHED kemajuan peluncuran

Saat Anda menerbitkan pembaruan ke `$LATEST.PUBLISHED`, Lambda mulai meluncurkan versi baru dalam langkah-langkah. Pada setiap langkah, Lambda menciptakan lingkungan eksekusi baru — hingga 20 persen dari total versi lama — dan secara bertahap menggantikan lingkungan eksekusi versi lama.

Dua faktor menentukan berapa lama peluncuran berlangsung:

- Waktu inisialisasi - Setiap langkah menunggu lingkungan eksekusi baru untuk diinisialisasi sebelum dilanjutkan. Jika inisialisasi memakan waktu 10 menit, setiap langkah menunggu setidaknya 10 menit.
- Memanggil durasi - Sebelum Lambda dapat menghapus lingkungan eksekusi lama, pemanggilan dalam penerbangannya harus selesai (drain). Jika durasi pemanggilan rata-rata Anda adalah 5 menit, pengeringan rata-rata memakan waktu sekitar 5 menit pada setiap langkah.

Karena Instans Terkelola Lambda menjaga pemanfaatan instans tetap tinggi, inisialisasi dan pengeringan berkontribusi pada total waktu penerapan. Fungsi dengan inisialisasi yang lama atau pemanggilan yang berjalan lama membutuhkan waktu lebih lama untuk diluncurkan sepenuhnya.

Bagaimana versi numerik dengan alias berkembang

Pembaruan alias tertimbang dilakukan secara bertahap dengan cara yang sama. Saat Anda mengubah bobot routing pada alias—misalnya, dari 90 persen versi 1 dan 10 persen versi 2 menjadi 100 persen versi 2— Instans Terkelola Lambda secara perlahan menggantikan lingkungan eksekusi versi 1 dengan versi 2, tanpa menyediakan kapasitas instans berlebih.

Gunakan versi numerik dengan alias ketika Anda ingin kontrol yang lebih ketat atas peluncuran bertahap dan pengamatan khusus versi. Karena setiap versi adalah kualifikasi yang berbeda, Anda dapat membandingkan metrik per versi untuk burung kenari dan versi stabil sebelum Anda mempromosikan perubahan. Untuk informasi lebih lanjut, lihat melihat metrik untuk fungsi Lambda.

Mengkonfigurasi penskalaan setelah penerapan

Setiap versi fungsi yang Anda terapkan memiliki konfigurasi penskalaannya sendiri, yang mengontrol jumlah minimum dan maksimum lingkungan eksekusi yang dipertahankan Lambda untuk versi tersebut. Tetapkan konfigurasi penskalaan setelah Anda menerbitkan versi sehingga versi baru dapat melayani lalu lintas pada kapasitas yang Anda harapkan.

Contoh berikut menetapkan konfigurasi penskalaan untuk versi tertentu. Ganti 2 dengan nomor versi Anda atau dengan `$LATEST.PUBLISHED`.

```
aws lambda put-function-scaling-config \
  --function-name my-managed-instance-function \
  --qualifier 2 \
  --function-scaling-config MinExecutionEnvironments=3,MaxExecutionEnvironments=20
```

Mengelola penerapan dengan infrastruktur sebagai kode

Anda dapat menentukan dan menerapkan fungsi Instans Terkelola Lambda dan penyedia kapasitasnya dengan infrastruktur sebagai kode, termasuk AWS CloudFormation, AWS Kit Pengembangan Cloud (AWS CDK), dan alat pihak ketiga seperti HashiCorp Terraform. Model penyedia kapasitas, fungsi, versi, dan alias Anda sebagai sumber daya sehingga strategi penerapan Anda dapat direproduksi dan dapat ditinjau.

Ingatlah pertimbangan berikut saat mengelola penerapan Instans Terkelola Lambda dengan IaC:

- Publikasikan versi untuk mengaktifkan perubahan. Versi fungsi baru menjadi aktif pada instans penyedia kapasitas hanya setelah dipublikasikan. Model setiap versi yang ingin Anda pertahankan dalam layanan—misalnya, kedua sisi blue/green shift—sebagai sumber dayanya sendiri.
- Konfigurasi penskalaan adalah per versi. Terapkan pengaturan lingkungan eksekusi minimum dan maksimum untuk setiap versi atau ke `$LATEST.PUBLISHED`. Jika alat iAC Anda tidak memodelkan konfigurasi penskalaan secara langsung, atur dengan `put-function-scaling-config` AWS Perintah CLI setelah penerapan.
- Paket penerapan harus berasal dari Amazon S3 saat Anda menggunakan AWS API Kontrol Awan. Yang AWS Cloud Control API menerima kode fungsi hanya sebagai lokasi Amazon S3

atau URI gambar kontainer, bukan sebagai file.zip yang diunggah secara lokal. Jika alat iAC Anda menggunakan API ini, unggah paket penerapan Anda ke bucket S3 terlebih dahulu dan referensikan dengan bucket dan kunci. Gunakan bucket berversi sehingga pembaruan pada objek terdeteksi.

Contoh: CloudFormation templat

Berikut ini CloudFormation Template menunjukkan a blue/green penyebaran. Ini menciptakan penyedia kapasitas, fungsi Lambda Managed Instances, dua versi yang diterbitkan (masing-masing dengan konfigurasi penskalaan fungsinya sendiri), dan alias bernama `production` yang mengarahkan 90 persen lalu lintas ke versi stabil dan 10 persen ke versi kenari. Untuk mempromosikan kenari, atur alias `FunctionVersion` ke versi kenari dan hapus `RoutingConfig`. Untuk memutar kembali, lepaskan `RoutingConfig` sehingga semua lalu lintas kembali ke versi stabil. Ganti nilai subnet, grup keamanan, peran, dan Amazon S3 dengan nilai Anda sendiri.

```
AWSTemplateFormatVersion: '2010-09-09'
Description: Lambda Managed Instances blue/green deployment with a weighted alias

Resources:
  MyCapacityProvider:
    Type: AWS::Lambda::CapacityProvider
    Properties:
      CapacityProviderName: my-capacity-provider
      VpcConfig:
        SubnetIds:
          - subnet-0123456789abcdef0
        SecurityGroupIds:
          - sg-0123456789abcdef0
      PermissionsConfig:
        CapacityProviderOperatorRoleArn: arn:aws:iam::111122223333:role/my-capacity-
provider-operator-role
      InstanceRequirements:
        Architectures:
          - arm64

  MyFunction:
    Type: AWS::Lambda::Function
    Properties:
      FunctionName: my-managed-instance-function
      Role: arn:aws:iam::111122223333:role/my-function-execution-role
      Runtime: python3.13
```

```
      Handler: index.lambda_handler
      Architectures:
        - arm64
      Code:
        S3Bucket: my-deployment-bucket
        S3Key: function.zip
      CapacityProviderConfig:
        LambdaManagedInstancesCapacityProviderConfig:
          CapacityProviderArn: !GetAtt MyCapacityProvider.Arn

  # Stable version - receives the majority of traffic
  StableVersion:
    Type: AWS::Lambda::Version
    Properties:
      FunctionName: !Ref MyFunction
      FunctionScalingConfig:
        MinExecutionEnvironments: 10
        MaxExecutionEnvironments: 50

  # Canary version - receives a small percentage of traffic
  CanaryVersion:
    Type: AWS::Lambda::Version
    Properties:
      FunctionName: !Ref MyFunction
      FunctionScalingConfig:
        MinExecutionEnvironments: 3
        MaxExecutionEnvironments: 20

  ProductionAlias:
    Type: AWS::Lambda::Alias
    Properties:
      FunctionName: !Ref MyFunction
      Name: production
      FunctionVersion: !GetAtt StableVersion.Version
      RoutingConfig:
        AdditionalVersionWeights:
          - FunctionVersion: !GetAtt CanaryVersion.Version
            FunctionWeight: 0.1
```

Contoh: CDK

Contoh CDK berikut (TypeScript) mendefinisikan penyedia kapasitas dan fungsi yang sama menggunakan konstruksi level 1 (L1) dari `aws-cdk-lib/aws-lambda` modul. `operatorRole` dan `executionRole` apakah peran IAM didefinisikan di tempat lain di tumpukan Anda.

```
import * as cdk from 'aws-cdk-lib';
import * as lambda from 'aws-cdk-lib/aws-lambda';

const capacityProvider = new lambda.CfnCapacityProvider(this, 'MyCapacityProvider', {
  capacityProviderName: 'my-capacity-provider',
  vpcConfig: {
    subnetIds: ['subnet-0123456789abcdef0'],
    securityGroupIds: ['sg-0123456789abcdef0'],
  },
  permissionsConfig: {
    capacityProviderOperatorRoleArn: operatorRole.roleArn,
  },
  instanceRequirements: {
    architectures: ['arm64'],
  },
});

const fn = new lambda.CfnFunction(this, 'MyFunction', {
  functionName: 'my-managed-instance-function',
  role: executionRole.roleArn,
  runtime: 'python3.13',
  handler: 'index.lambda_handler',
  architectures: ['arm64'],
  code: { s3Bucket: 'my-deployment-bucket', s3Key: 'function.zip' },
  capacityProviderConfig: {
    lambdaManagedInstancesCapacityProviderConfig: {
      capacityProviderArn: capacityProvider.attrArn,
    },
  },
  functionScalingConfig: {
    minExecutionEnvironments: 3,
    maxExecutionEnvironments: 20,
  },
  publishToLatestPublished: true,
});
fn.addDependency(capacityProvider);
```

Contoh: penyebaran kenari dengan AWS CodeDeploy

Contoh CDK berikut (TypeScript) menggunakan AWS CodeDeploy untuk mengalihkan lalu lintas ke versi fungsi baru secara otomatis. Yang `LambdaDeploymentGroup` mengirimkan 10 persen lalu lintas ke versi baru selama lima menit dan kemudian menggeser sisanya. Jika CloudWatch alarm memasuki keadaan alarm selama penerapan, CodeDeploy memutar kembali ke versi sebelumnya.

`fn` adalah L1 `CfnFunction` dari contoh sebelumnya. Karena CodeDeployadalah `LambdaDeploymentGroup` bekerja dengan alias level 2 (L2) dan konstruksi fungsi, impor fungsi L1 oleh ARN dan buat versi dan alias darinya.

```
import * as cdk from 'aws-cdk-lib';
import * as lambda from 'aws-cdk-lib/aws-lambda';
import * as codedeploy from 'aws-cdk-lib/aws-codedeploy';
import * as cloudwatch from 'aws-cdk-lib/aws-cloudwatch';

// Import the L1 function so that L2 constructs can reference it.
const importedFn = lambda.Function.fromFunctionArn(this, 'ImportedFunction',
 fn.attrArn);

// Publish the version that CodeDeploy shifts traffic to.
const canaryVersion = new lambda.CfnVersion(this, 'CanaryVersion', {
  functionName: fn.ref,
});

// Create the alias that CodeDeploy shifts traffic on.
const alias = new lambda.Alias(this, 'ProductionAlias', {
  aliasName: 'production',
  version: lambda.Version.fromVersionAttributes(this, 'CanaryVersionRef', {
    lambda: importedFn,
    version: canaryVersion.attrVersion,
  }),
});

// Roll back automatically if this alarm fires during the deployment.
const errorAlarm = new cloudwatch.Alarm(this, 'ErrorAlarm', {
  metric: importedFn.metricErrors({ period: cdk.Duration.minutes(1) }),
  threshold: 5,
  evaluationPeriods: 1,
  comparisonOperator: cloudwatch.ComparisonOperator.GREATER_THAN_OR_EQUAL_TO_THRESHOLD,
});

new codedeploy.LambdaDeploymentGroup(this, 'CanaryDeployment', {
```

```
  alias,
  deploymentConfig: codedeploy.LambdaDeploymentConfig.CANARY_10PERCENT_5MINUTES,
  alarms: [errorAlarm],
  autoRollback: {
    failedDeployment: true,
    deploymentInAlarm: true,
  },
});
```

Langkah selanjutnya

- Memahami penskalaan untuk Instans Terkelola Lambda
- Pelajari tentang penyedia kapasitas untuk Instans Terkelola Lambda
- Tinjauan praktik terbaik untuk Instans Terkelola Lambda

Waktu proses Instans Terkelola Lambda

Fungsi Peristiwa Lambda pada Instans Terkelola Lambda mendukung keduanya `.zip` arsip file dan paket penyebaran gambar kontainer, yang menggunakan format Open Container Initiative (OCI). Untuk mengurangi waktu startup untuk gambar kontainer besar, lihat Lazy-load gambar kontainer dengan Seekable OCI (SOCI).

Lambda memproses permintaan secara berbeda saat menggunakan Lambda Managed Instances. Alih-alih menangani permintaan secara berurutan di setiap lingkungan eksekusi, Lambda Managed Instances memproses beberapa permintaan secara bersamaan dalam setiap lingkungan eksekusi. Perubahan dalam model eksekusi ini berarti bahwa Lambda Event Functions yang menggunakan Lambda Managed Instances perlu mempertimbangkan keamanan thread, manajemen status, dan isolasi konteks. Kekhawatiran ini tidak muncul dalam model konkurensi tunggal Lambda (default). Selain itu, implementasi multi-konkurensi bervariasi antara runtime.

Bahasa yang didukung

Instans Terkelola Lambda dapat digunakan dengan bahasa pemrograman dan runtime berikut:

- Jawa: Java 21 dan seterusnya.
- Python: Python 3.13 dan yang lebih baru.
- Node.js: Node.js 22 dan seterusnya.
- .NET: .NET 8 dan yang lebih baru.

- Karat: Didukung menggunakan OS-only runtime `provided.al2023` dan kemudian.

Language-specific pertimbangan

Setiap bahasa pemrograman mengimplementasikan multi-concurrency secara berbeda. Anda perlu memahami bagaimana multi-konkurensi diimplementasikan dalam bahasa pemrograman pilihan Anda untuk menerapkan praktik terbaik konkurensi yang sesuai.

Java

Menggunakan proses tunggal dengan utas OS untuk konkurensi. Beberapa utas mengeksekusi metode handler secara bersamaan, membutuhkan penanganan status dan sumber daya bersama yang aman dari thread.

Python

Menggunakan beberapa proses Python di mana setiap permintaan bersamaan berjalan dalam proses terpisah. Ini melindungi dari sebagian besar masalah konkurensi, meskipun perawatan diperlukan untuk sumber daya bersama seperti `/tmp` direktori.

Node.js

Kegunaan utas pekerja dengan eksekusi asinkron. Permintaan bersamaan didistribusikan di seluruh thread pekerja, dan setiap thread pekerja juga dapat menangani permintaan bersamaan secara asinkron, membutuhkan penanganan status dan sumber daya bersama yang aman.

.NET

Menggunakan Tugas.NET dengan pemrosesan asinkron dari beberapa permintaan bersamaan. Membutuhkan penanganan yang aman dari sumber daya negara dan bersama.

Karat

Menggunakan satu proses dengan tugas asinkron yang didukung oleh Tokyo. Handler harus `Clone + Send`.

Langkah selanjutnya

Untuk informasi terperinci tentang setiap runtime, lihat topik berikut:

- Runtime Java untuk Instans Terkelola Lambda
- Node.js runtime untuk Instans Terkelola Lambda

- Runtime Python untuk Instans Terkelola Lambda
- Runtime.NET untuk Instans Terkelola Lambda
- Dukungan Rust untuk Instans Terkelola Lambda

Runtime Java untuk Instans Terkelola Lambda

Untuk runtime Java, Lambda Managed Instances menggunakan utas OS untuk konkurensi. Lambda memuat objek handler Anda sekali per lingkungan eksekusi selama inisialisasi dan kemudian membuat beberapa utas. Thread ini dijalankan secara paralel dan memerlukan penanganan status dan sumber daya bersama yang aman dari thread. Setiap thread berbagi objek handler yang sama dan bidang statis apa pun.

Konfigurasi konkurensi

Jumlah maksimum permintaan bersamaan yang dikirim Lambda ke setiap lingkungan eksekusi dikendalikan oleh `PerExecutionEnvironmentMaxConcurrency` pengaturan dalam konfigurasi fungsi. Ini adalah pengaturan opsional, dan nilai default bervariasi tergantung pada runtime. Untuk runtime Java, defaultnya adalah 32 permintaan bersamaan per vCPU, atau Anda dapat mengonfigurasi nilai Anda sendiri. Nilai ini juga menentukan jumlah thread yang digunakan oleh runtime Java. Lambda secara otomatis menyesuaikan jumlah permintaan bersamaan hingga maksimum yang dikonfigurasi berdasarkan kapasitas setiap lingkungan eksekusi untuk menyerap permintaan tersebut.

Membangun fungsi untuk multi-konkurensi

Anda harus menerapkan praktik keamanan thread yang sama saat menggunakan Instans Terkelola Lambda seperti yang Anda lakukan di lingkungan multi-utas lainnya. Karena objek handler dibagikan di semua utas pekerja runtime, setiap status yang dapat berubah harus aman untuk utas. Ini termasuk koleksi, koneksi database, dan objek statis apa pun yang dimodifikasi selama pemrosesan permintaan.

AWS Klien SDK aman untuk thread dan tidak memerlukan penanganan khusus.

Contoh: Kumpulan koneksi database

Kode berikut menggunakan objek koneksi database statis yang dibagi antara utas. Tergantung pada pustaka koneksi yang digunakan, ini mungkin tidak aman utas.

```
public class DBQueryHandler implements RequestHandler<Object, String> {
```

```
    // Single connection shared across all threads - NOT SAFE
    private static Connection connection;

    public DBQueryHandler() {
        this.connection = DriverManager.getConnection(jdbcUrl, username, password);
    }

    @Override
    public String handleRequest(Object input, Context context) {
        PreparedStatement stmt = connection.prepareStatement(query);
        ResultSet rs = stmt.executeQuery();
        // Multiple threads using same connection causes issues
        return result.toString();
    }
}
```

Pendekatan Thread-safe adalah dengan menggunakan kumpulan koneksi. Dalam contoh berikut, pengendali fungsi mengambil koneksi dari kumpulan. Koneksi hanya digunakan dalam konteks permintaan tunggal.

```
public class DBQueryHandler implements RequestHandler<Object, String> {

    private static HikariDataSource dataSource;

    public DBQueryHandler() {
        HikariConfig config = new HikariConfig();
        config.setJdbcUrl("jdbc:mysql://localhost:3306/your_database");
        dataSource = new HikariDataSource(config); // Create pool once per Lambda
 container
    }

    @Override
    public String handleRequest(Object input, Context context) {
        String query = "SELECT column_name FROM your_table LIMIT 10";
        StringBuilder result = new StringBuilder("Data:\n");

        // try-with-resources automatically calls close() on the connection,
        // which returns it to the HikariCP pool (does NOT close the physical DB
 connection)
        try (Connection connection = dataSource.getConnection();
             PreparedStatement stmt = connection.prepareStatement(query);
             ResultSet rs = stmt.executeQuery()) {
```

```
            while (rs.next()) {
                result.append(rs.getString("column_name")).append("\n");
            }

        } catch (Exception e) {
            context.getLogger().log("Error: " + e.getMessage());
            return "Error";
        }

        return result.toString();
    }
}
```

Contoh: Koleksi

Koleksi Java standar tidak aman utas:

```
public class Handler implements RequestHandler<Object, String> {
    private static List<String> items = new ArrayList<>();
    private static Map<String, Object> cache = new HashMap<>();

    @Override
    public String handleRequest(Object input, Context context) {
        items.add("list item");  // Not thread-safe
        cache.put("key", input); // Not thread-safe
        return "Success";
    }
}
```

Sebagai gantinya, gunakan koleksi Thread-safe:

```
public class Handler implements RequestHandler<Object, String> {
    private static final List<String> items =
        Collections.synchronizedList(new ArrayList<>());
    private static final ConcurrentHashMap<String, Object> cache =
        new ConcurrentHashMap<>();

    @Override
    public String handleRequest(Object input, Context context) {
        items.add("list item");  // Thread-safe
        cache.put("key", input); // Thread-safe
        return "Success";
    }
```

```
}
```

Direktori /tmp yang dibagikan

Direk `/tmp` tori dibagikan di semua permintaan bersamaan di lingkungan eksekusi. Penulisan bersamaan ke file yang sama dapat menyebabkan kerusakan data, misalnya jika proses lain mengganti file. Untuk mengatasi hal ini, terapkan penguncian file untuk file bersama atau gunakan nama file unik per utas atau per permintaan untuk menghindari konflik. Ingatlah untuk membersihkan file yang tidak dibutuhkan untuk menghindari kehabisan ruang yang tersedia.

Pencatatan

Interleaving log (entri log dari permintaan berbeda yang disisipkan dalam log) normal dalam sistem multi-concurrent.

Fungsi yang menggunakan Instans Terkelola Lambda selalu menggunakan format log JSON terstruktur yang diperkenalkan dengan kontrol logging lanjutan. Format ini mencakup`requestId`, memungkinkan entri log untuk dikorelasikan dengan satu permintaan. Ketika Anda menggunakan `LambdaLogger` objek dari `context.getLogger()` secara otomatis `requestId` disertakan dalam setiap entri log. Untuk informasi lebih lanjut, lihatthe section called “Menggunakan kontrol logging lanjutan Lambda dengan Java”.

Permintaan konteks

Ob `context` jek terikat ke thread permintaan. Menggunakan `context.getAwsRequestId()` menyediakan akses Thread-safe ke ID permintaan untuk permintaan saat ini.

Gunakan `context.getXrayTraceId()` untuk mengakses ID X-Ray jejak. Ini menyediakan akses aman utas ke ID jejak untuk permintaan saat ini. Lambda tidak mendukung variabel `_X_AMZN_TRACE_ID` lingkungan dengan Lambda Managed Instances. ID X-Ray jejak disebarkan secara otomatis saat menggunakan AWS SDK.

Gunakan `com.amazonaws.services.lambda.runtime.Context.getRemainingTimeInMillis()` untuk mendeteksi batas waktu. Untuk informasi selengkapnya, lihat the section called “Penanganan dan pemulihan kesalahan”.

Jika Anda menggunakan utas virtual dalam program Anda atau membuat utas selama inisialisasi, Anda perlu meneruskan konteks permintaan yang diperlukan ke utas ini.

Contoh: Penanganan batas waktu

Periksa sisa waktu sebelum setiap unit kerja dan hentikan pemrosesan sebelum batas waktu dimulai. Konfigur BUFFER_MS asikan berdasarkan durasi yang diharapkan dari potongan pekerjaan Anda berikutnya.

```
private static final int BUFFER_MS = 2000; // Configure based on your next chunk of
 work

public Map<String, Object> handleRequest(Map<String, Object> event, Context context) {
    for (Object item : (List<Object>) event.get("items")) {
        if (context.getRemainingTimeInMillis() < BUFFER_MS)
            return Map.of("statusCode", 206, "body", "Timeout approaching, stopping
 early");
        processItem(item);
    }
    return Map.of("statusCode", 200, "body", "Done");
}
```

Contoh: Menyebarkan tenggat waktu ke panggilan hilir

Saat melakukan panggilan ke layanan hilir, sebarkan waktu yang tersisa sebagai batas waktu untuk menghindari ketergantungan pada panggilan jaringan yang akan bertahan lebih lama dari panggilan Anda. Gunakan penggantian per permintaan pada klien bersama daripada membuat klien baru per pemanggilan:

```
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.model.GetObjectRequest;
import java.time.Duration;

private static final S3Client s3 = S3Client.create();

public String handleRequest(Map<String, Object> event, Context context) {
    Duration timeout = Duration.ofMillis(Math.max(1000,
 context.getRemainingTimeInMillis() - 500));

    GetObjectRequest req = GetObjectRequest.builder()
        .bucket("my-bucket").key("my-key")
        .overrideConfiguration(cfg -> cfg.apiCallTimeout(timeout))
        .build();
    s3.getObject(req);
    return "Done";
```

```
}
```

Inisialisasi dan shutdown

Inisialisasi fungsi terjadi sekali per lingkungan eksekusi. Objek yang dibuat selama inisialisasi dibagikan di seluruh thread.

Untuk fungsi Lambda dengan ekstensi, lingkungan eksekusi memancarkan sinyal SIGTERM selama shutdown. Sinyal ini digunakan oleh ekstensi untuk memicu tugas pembersihan, seperti buffer pembilasan. Anda dapat berlangganan peristiwa SIGTERM untuk memicu tugas pembersihan fungsi, seperti menutup koneksi database. Untuk mempelajari lebih lanjut tentang siklus hidup lingkungan eksekusi, lihatthe section called “Lingkungan eksekusi”.

Versi ketergantungan

Instans Terkelola Lambda memerlukan versi paket minimum berikut:

- AWS SDK untuk Java 2.0: versi 2.34.0 atau yang lebih baru
- AWS X-Ray SDK untuk Java: versi 2.20.0 atau yang lebih baru
- AWS Distro untuk OpenTelemetry - Instrumentasi untuk Java: versi 2.20.0 atau yang lebih baru
- Powertools untuk AWS Lambda (Java): versi 2.8.0 atau yang lebih baru

Powertools untuk AWS Lambda (Jawa)

Powertools untuk AWS Lambda (Java) kompatibel dengan Lambda Managed Instances dan menyediakan utilitas untuk logging, pelacakan, metrik, dan banyak lagi. Untuk informasi selengkapnya, lihat Powertools untuk AWS Lambda (Java).

Langkah selanjutnya

- Tinjau Node.js runtime untuk Instans Terkelola Lambda
- Tinjau runtime Python untuk Instans Terkelola Lambda
- Tinjau runtime .NET untuk Instans Terkelola Lambda
- Pelajari tentang penskalaan Instans Terkelola Lambda

Node.js runtime untuk Instans Terkelola Lambda

Untuk Node.js runtime, Lambda Managed Instances menggunakan thread pekerja dengan eksekusi `await` berbasis`async`/untuk menangani permintaan bersamaan. Inisialisasi fungsi terjadi sekali

per thread pekerja. Pemanggilan bersamaan ditangani di dua dimensi: utas pekerja menyediakan paralelisme di seluruh vCPU, dan eksekusi asinkron menyediakan konkurensi dalam setiap utas. Setiap permintaan bersamaan yang ditangani oleh thread pekerja yang sama berbagi objek handler dan status global yang sama, membutuhkan penanganan yang aman di bawah beberapa permintaan bersamaan.

Konkurensi maksimum

Jumlah maksimum permintaan bersamaan yang dikirim Lambda ke setiap lingkungan eksekusi dikendalikan oleh `PerExecutionEnvironmentMaxConcurrency` pengaturan dalam konfigurasi fungsi. Ini adalah pengaturan opsional, dan nilai default bervariasi tergantung pada runtime. Untuk Node.js runtime, defaultnya adalah 64 permintaan bersamaan per vCPU, atau Anda dapat mengonfigurasi nilai Anda sendiri. Lambda secara otomatis menyesuaikan jumlah permintaan bersamaan hingga maksimum yang dikonfigurasi berdasarkan kapasitas setiap lingkungan eksekusi untuk menyerap permintaan tersebut.

Untuk Node.js, jumlah permintaan bersamaan yang dapat diproses oleh setiap lingkungan eksekusi ditentukan oleh jumlah utas pekerja dan kapasitas setiap utas pekerja untuk memproses permintaan bersamaan secara asinkron. Jumlah default utas pekerja ditentukan oleh jumlah vCPU yang tersedia, atau Anda dapat mengonfigurasi jumlah utas pekerja dengan menyetel variabel `AWS_LAMBDA_NODEJS_WORKER_COUNT` lingkungan. Sebaiknya gunakan penangan fungsi async karena ini memungkinkan pemrosesan beberapa permintaan per utas pekerja. Jika pengendali fungsi Anda sinkron, setiap thread pekerja hanya dapat memproses satu permintaan pada satu waktu.

Membangun fungsi untuk multi-konkurensi

Dengan pengendali fungsi async, setiap pekerja runtime memproses beberapa permintaan secara bersamaan. Objek global dibagikan di beberapa permintaan bersamaan. Untuk objek yang dapat berubah, hindari menggunakan status atau penggunaan `AsyncLocalStorage` global.

AWS Klien SDK aman asinkron dan tidak memerlukan penanganan khusus.

Contoh: Negara global

Kode berikut menggunakan objek global yang bermutasi di dalam handler fungsi. Ini tidak aman asinkron.

```
let state = {
    currentUser: null,
```

```
    requestData: null
};

export const handler = async (event, context) => {
    state.currentUser = event.userId;
    state.requestData = event.data;

    await processData(state.requestData);

    // state.currentUser might now belong to a different request
    return { user: state.currentUser };
};
```

Menginisialisasi `state` objek di dalam pengendali fungsi menghindari status global bersama.

```
export const handler = async (event, context) => {
    let state = {
        currentUser: event.userId,
        requestData: event.data
    };

    await processData(state.requestData);

    return { user: state.currentUser };
};
```

Contoh: Koneksi database

Kode berikut menggunakan objek klien bersama yang dibagi antara beberapa pemanggilan. Tergantung pada pustaka koneksi yang digunakan, ini mungkin tidak aman untuk konkurensi.

```
const { Client } = require('pg');

// Single connection created at init time
const client = new Client({
  host: process.env.DB_HOST,
  database: process.env.DB_NAME,
  user: process.env.DB_USER,
  password: process.env.DB_PASSWORD
});

// Connect once during cold start
client.connect();
```

```
exports.handler = async (event) => {
  // Multiple parallel invocations share this single connection = BAD
  // With multi-concurrent Lambda, queries will collide
  const result = await client.query('SELECT * FROM users WHERE id = $1',
 [event.userId]);

  return {
    statusCode: 200,
    body: JSON.stringify(result.rows[0])
  };
};
```

Pendekatan yang aman untuk bersaing adalah dengan menggunakan kumpulan koneksi. Kumpulan menggunakan koneksi terpisah untuk setiap kueri database bersamaan.

```
const { Pool } = require('pg');

// Connection pool created at init time
const pool = new Pool({
  host: process.env.DB_HOST,
  database: process.env.DB_NAME,
  user: process.env.DB_USER,
  password: process.env.DB_PASSWORD,
  max: 20,  // Max connections in pool
  idleTimeoutMillis: 30000,
  connectionTimeoutMillis: 2000
});

exports.handler = async (event) => {
  // Pool gives each parallel invocation its own connection
  const result = await pool.query('SELECT * FROM users WHERE id = $1', [event.userId]);

  return {
    statusCode: 200,
    body: JSON.stringify(result.rows[0])
  };
};
```

Node.js 22 penangan berbasis panggilan balik

Saat menggunakan Node.js 22, Anda tidak dapat menggunakan pengendali fungsi berbasis panggilan balik dengan Instans Terkelola Lambda. Callback-based penangan hanya didukung untuk

fungsi Lambda (default). Untuk runtime Node.js 24 dan yang lebih baru, penangan fungsi berbasis callback tidak digunakan lagi untuk Lambda (default) dan Lambda Managed Instances.

Sebagai gantinya, gunakan pengen `async` dali fungsi saat menggunakan Lambda Managed Instances. Untuk informasi selengkapnya, lihat Men definisikan pengendali fungsi Lambda di Node.js.

Direktori /tmp yang dibagikan

Direk `/tmp` tori dibagikan di semua permintaan bersamaan di lingkungan eksekusi. Penulisan bersamaan ke file yang sama dapat menyebabkan kerusakan data, misalnya jika proses lain mengganti file. Untuk mengatasi hal ini, terapkan penguncian file untuk file bersama atau gunakan nama file unik per permintaan untuk menghindari konflik. Ingatlah untuk membersihkan file yang tidak dibutuhkan untuk menghindari kehabisan ruang yang tersedia.

Pencatatan

Interleaving log (entri log dari permintaan berbeda yang disisipkan dalam log) normal dalam sistem multi-concurrent. Fungsi yang menggunakan Instans Terkelola Lambda selalu menggunakan format log JSON terstruktur yang diperkenalkan dengan kontrol logging lanjutan. Format ini mencakup`requestId`, memungkinkan entri log untuk dikorelasikan dengan satu permintaan. Ketika Anda menggunakan `console` logger, secara otomatis `requestId` disertakan dalam setiap entri log. Untuk informasi lebih lanjut, lihatthe section called “Menggunakan kontrol logging lanjutan Lambda dengan Node.js”.

Pustaka logging pihak ketiga yang populer, seperti Winston, biasanya menyertakan dukungan untuk menggunakan konsol untuk keluaran log.

Permintaan konteks

Menggunakan `context.awsRequestId` menyediakan akses async-safe ke ID permintaan untuk permintaan saat ini.

Gunakan `context.xRayTraceId` untuk mengakses ID X-Ray jejak. Ini menyediakan akses aman secara bersamaan ke ID jejak untuk permintaan saat ini. Lambda tidak mendukung variabel `_X_AMZN_TRACE_ID` lingkungan dengan Lambda Managed Instances. ID X-Ray jejak disebarkan secara otomatis saat menggunakan AWS SDK.

Gunakan `context.getRemainingTimeInMillis()` untuk mendeteksi batas waktu. Untuk informasi selengkapnya, lihat the section called “Penanganan dan pemulihan kesalahan”.

Contoh: Penanganan batas waktu

Periksa sisa waktu sebelum setiap unit kerja dan hentikan pemrosesan sebelum batas waktu dimulai. Konfigur BUFFER_MS asikan berdasarkan durasi yang diharapkan dari potongan pekerjaan Anda berikutnya.

```
const BUFFER_MS = 2000; // Configure based on your next chunk of work

exports.handler = async (event, context) => {
    for (const item of event.items) {
        if (context.getRemainingTimeInMillis() < BUFFER_MS)
            return { statusCode: 206, body: "Timeout approaching, stopping early" };
        await processItem(item);
    }
    return { statusCode: 200, body: "Done" };
};
```

Contoh: Menyebarkan tenggat waktu ke panggilan hilir

Saat melakukan panggilan ke layanan hilir, sebarkan waktu yang tersisa sebagai batas waktu untuk menghindari ketergantungan pada panggilan jaringan yang akan bertahan lebih lama dari panggilan Anda:

```
const { S3Client, GetObjectCommand } = require("@aws-sdk/client-s3");
const client = new S3Client({});

exports.handler = async (event, context) => {
    const timeout = Math.max(1000, context.getRemainingTimeInMillis() - 500);
    const response = await client.send(
        new GetObjectCommand({ Bucket: "my-bucket", Key: "my-key" }),
        { abortSignal: AbortSignal.timeout(timeout) }
    );
    return { statusCode: 200, body: "Done" };
};
```

Inisialisasi dan shutdown

Inisialisasi fungsi terjadi sekali per thread pekerja. Anda mungkin melihat entri log berulang jika fungsi Anda memancarkan log selama inisialisasi.

Untuk fungsi Lambda dengan ekstensi, lingkungan eksekusi memancarkan sinyal SIGTERM selama shutdown. Sinyal ini digunakan oleh ekstensi untuk memicu tugas pembersihan, seperti buffer pembilasan. Fungsi Lambda (default) dengan ekstensi juga dapat berlangganan sinyal SIGTERM menggunakan. `process.on()` Ini tidak didukung untuk fungsi yang menggunakan Instans Terkelola

Lambda karena `process.on()` tidak dapat digunakan dengan utas pekerja. Untuk mempelajari lebih lanjut tentang siklus hidup lingkungan eksekusi, lihatthe section called “Lingkungan eksekusi”.

Versi ketergantungan

Instans Terkelola Lambda memerlukan versi paket minimum berikut:

- AWS SDK untuk JavaScript v3: versi 3.933.0 atau yang lebih baru
- AWS X-Ray SDK untuk Node.js: versi 3.12.0 atau yang lebih baru
- AWS Distro untuk OpenTelemetry - Instrumentasi untuk JavaScript: versi 0.8.0 atau yang lebih baru
- Powertools untuk AWS Lambda (TypeScript): versi 2.29.0 atau yang lebih baru

Powertools untuk AWS Lambda () TypeScript

Powertools for AWS Lambda (TypeScript) kompatibel dengan Lambda Managed Instances dan menyediakan utilitas untuk logging, pelacakan, metrik, dan banyak lagi. Untuk informasi selengkapnya, lihat Powertools for AWS Lambda ()TypeScript.

Langkah selanjutnya

- Tinjau runtime Java untuk Instans Terkelola Lambda
- Tinjau runtime Python untuk Instans Terkelola Lambda
- Tinjau runtime .NET untuk Instans Terkelola Lambda
- Pelajari tentang penskalaan Instans Terkelola Lambda

Runtime Python untuk Instans Terkelola Lambda

Lambda runtime menggunakan beberapa proses Python untuk menangani permintaan bersamaan. Setiap permintaan bersamaan berjalan dalam proses terpisah dengan ruang memori dan inisialisasi sendiri. Setiap proses menangani satu permintaan pada satu waktu, secara sinkron. Proses tidak berbagi memori secara langsung, sehingga variabel global, cache tingkat modul, dan objek tunggal diisolasi di antara permintaan bersamaan.

Konfigurasi konkurensi

Jumlah maksimum permintaan bersamaan yang dikirim Lambda ke setiap lingkungan eksekusi dikendalikan oleh `PerExecutionEnvironmentMaxConcurrency` pengaturan dalam konfigurasi

fungsi. Ini adalah pengaturan opsional, dan nilai default bervariasi tergantung pada runtime. Untuk runtime Python, defaultnya adalah 16 permintaan bersamaan per vCPU, atau Anda dapat mengonfigurasi nilai Anda sendiri. Nilai ini juga menentukan jumlah proses yang digunakan oleh runtime Python. Lambda secara otomatis menyesuaikan jumlah permintaan bersamaan hingga maksimum yang dikonfigurasi berdasarkan kapasitas setiap lingkungan eksekusi untuk menyerap permintaan tersebut.

> **Important**
>
> Menggunakan konkurensi berbasis proses berarti setiap proses runtime worker melakukan inisialisasi sendiri. Total penggunaan memori sama dengan memori per proses dikalikan dengan jumlah proses bersamaan. Jika Anda memuat perpustakaan besar atau kumpulan data dan memiliki konkurensi tinggi, Anda memiliki jejak memori yang besar. Sesuai dengan beban kerja Anda, Anda mungkin perlu menyesuaikan CPU-to-memory rasio atau menggunakan pengaturan konkurensi yang lebih rendah untuk menghindari melebihi memori yang tersedia. Anda dapat menggunakan `MemoryUtilization` metrik CloudWatch untuk melacak konsumsi memori.

Membangun fungsi untuk multi-konkurensi

Karena model multi-konkurensi berbasis proses, fungsi Instans Terkelola Lambda menggunakan runtime Python tidak mengakses sumber daya dalam memori secara bersamaan dari beberapa panggilan. Anda tidak perlu menerapkan praktik pengkodean untuk keamanan konkurensi dalam memori.

Direktori /tmp bersama

Direk `/tmp` tori dibagikan di semua permintaan bersamaan di lingkungan eksekusi. Penulisan bersamaan ke file yang sama dapat menyebabkan kerusakan data, misalnya jika proses lain mengganti file. Untuk mengatasi hal ini, terapkan penguncian file untuk file bersama atau gunakan nama file unik per proses atau per permintaan untuk menghindari konflik. Ingatlah untuk membersihkan file yang tidak dibutuhkan untuk menghindari kehabisan ruang yang tersedia.

Pencatatan

Interleaving log (entri log dari permintaan berbeda yang disisipkan dalam log) normal dalam sistem multi-concurrent.

Fungsi yang menggunakan Instans Terkelola Lambda selalu menggunakan format log JSON terstruktur yang diperkenalkan dengan kontrol logging lanjutan. Format ini mencakup`requestId`, memungkinkan entri log untuk dikorelasikan dengan satu permintaan. Ketika Anda menggunakan `logging` modul dari pustaka standar Python di Lambda, secara otomatis `requestId` disertakan dalam setiap entri log. Untuk informasi lebih lanjut, lihat Menggunakan kontrol logging lanjutan Lambda dengan Python.

Permintaan konteks

Gunakan `context.aws_request_id` untuk mengakses ID permintaan untuk permintaan saat ini.

Dengan runtime Python, Anda dapat menggunakan variabel `_X_AMZN_TRACE_ID` lingkungan untuk mengakses ID X-Ray jejak dengan Lambda Managed Instances. ID X-Ray jejak disebarkan secara otomatis saat menggunakan AWS SDK.

Gunakan `context.get_remaining_time_in_millis()` untuk mendeteksi batas waktu. Untuk informasi selengkapnya, lihat the section called “Penanganan dan pemulihan kesalahan”.

Contoh: Penanganan batas waktu

Periksa sisa waktu sebelum setiap unit kerja dan hentikan pemrosesan sebelum batas waktu dimulai. Konfigur `BUFFER_MS` asikan berdasarkan durasi yang diharapkan dari potongan pekerjaan Anda berikutnya.

```
BUFFER_MS = 2000  # Configure based on your next chunk of work

def handler(event, context):
    for item in event["items"]:
        if context.get_remaining_time_in_millis() < BUFFER_MS:
            return {"statusCode": 206, "body": "Timeout approaching, stopping early"}
        process_item(item)
    return {"statusCode": 200, "body": "Done"}
```

Contoh: Menyebarkan tenggat waktu ke panggilan hilir

Saat melakukan panggilan ke layanan hilir, sebarkan waktu yang tersisa sebagai batas waktu untuk menghindari ketergantungan pada panggilan jaringan yang akan bertahan lebih lama dari panggilan Anda. SDK boto3 tidak mendukung batas waktu per permintaan pada klien yang ada, jadi Anda harus membuat klien dengan batas waktu yang diinginkan. Untuk fungsi throughput tinggi, evaluasi apakah batas waktu tetap yang dikonfigurasi pada inisialisasi lebih tepat daripada pembuatan klien per permintaan.

```
import boto3
from botocore.config import Config

def handler(event, context):
    remaining = context.get_remaining_time_in_millis() / 1000
    timeout = max(1, remaining - 0.5)
    s3 = boto3.client("s3", config=Config(read_timeout=timeout,
 connect_timeout=timeout))
    response = s3.get_object(Bucket="my-bucket", Key="my-key")
    return {"statusCode": 200, "body": "Done"}
```

Inisialisasi dan shutdown

Inisialisasi fungsi terjadi sekali per proses. Anda mungkin melihat entri log berulang jika fungsi Anda memancarkan log selama inisialisasi.

Untuk fungsi Lambda dengan ekstensi, lingkungan eksekusi memancarkan sinyal SIGTERM selama shutdown. Sinyal ini digunakan oleh ekstensi untuk memicu tugas pembersihan, seperti buffer pembilasan. Anda dapat berlangganan peristiwa SIGTERM untuk memicu tugas pembersihan fungsi, seperti menutup koneksi database. Untuk mempelajari lebih lanjut tentang siklus hidup lingkungan eksekusi, lihatthe section called “Lingkungan eksekusi”.

Versi ketergantungan

Instans Terkelola Lambda memerlukan versi paket minimum berikut:

- Powertools untuk AWS Lambda (Python): versi 3.23.0 atau yang lebih baru

Powertools untuk AWS Lambda (Python)

Powertools for AWS Lambda (Python) kompatibel dengan Lambda Managed Instances dan menyediakan utilitas untuk logging, pelacakan, metrik, dan banyak lagi. Untuk informasi selengkapnya, lihat Powertools untuk AWS Lambda (Python).

Langkah selanjutnya

- Tinjau runtime Java untuk Instans Terkelola Lambda
- Tinjau Node.js runtime untuk Instans Terkelola Lambda
- Tinjau runtime .NET untuk Instans Terkelola Lambda

- Pelajari tentang penskalaan Instans Terkelola Lambda

.NET runtime untuk Instans Terkelola Lambda

Untuk runtime.NET, Instans Terkelola Lambda menggunakan satu proses.NET per lingkungan eksekusi. Beberapa permintaan bersamaan diproses menggunakan .NET Tasks.

Konfigurasi konkurensi

Jumlah maksimum permintaan bersamaan yang dikirim Lambda ke setiap lingkungan eksekusi dikendalikan oleh pengaturan `PerExecutionEnvironmentMaxConcurrency` dalam konfigurasi fungsi. Ini adalah pengaturan opsional, dan nilai default bervariasi tergantung pada runtime. Untuk runtime.NET, defaultnya adalah 32 permintaan bersamaan per vCPU, atau Anda dapat mengonfigurasi nilai Anda sendiri. Lambda secara otomatis menyesuaikan jumlah permintaan bersamaan hingga maksimum yang dikonfigurasi berdasarkan kapasitas setiap lingkungan eksekusi untuk menyerap permintaan tersebut.

Membangun fungsi untuk multi-konkurensi

Anda harus menerapkan praktik keamanan konkurensi yang sama saat menggunakan Instans Terkelola Lambda seperti yang Anda lakukan di lingkungan multi-konkuren lainnya. Karena objek handler dibagikan di semua Tugas, status apa pun yang bisa berubah harus aman untuk utas. Ini termasuk koleksi, koneksi database, dan objek statis apa pun yang dimodifikasi selama pemrosesan permintaan.

AWS Klien SDK aman untuk utas dan tidak memerlukan penanganan khusus.

Contoh: Kolam koneksi database

Kode berikut menggunakan objek koneksi database statis yang dibagi antara permintaan bersamaan. `SqlConnection`Objeknya tidak aman untuk utas.

```
public class DBQueryHandler
{
    // Single connection shared across threads - NOT SAFE
    private SqlConnection connection;

    public DBQueryHandler()
    {
        connection = new SqlConnection("your-connection-string-here");
        connection.Open();
```

```
    }

    public string Handle(object input, ILambdaContext context)
    {
        using var cmd = connection.CreateCommand();
        cmd.CommandText = "SELECT ..."; // your query

        using var reader = cmd.ExecuteReader();

        ...
    }
}
```

Untuk mengatasinya, gunakan koneksi terpisah untuk setiap permintaan, yang diambil dari kumpulan koneksi. ADO.NET penyedia seperti `Microsoft.Data.SqlClient` secara otomatis mendukung penyatuan koneksi ketika objek koneksi dibuka.

```
public class DBQueryHandler
{
    public DBQueryHandler()
    {
    }

    public string Handle(object input, ILambdaContext context)
    {
        using var connection = new SqlConnection("your-connection-string-here");
        connection.Open();
        using var cmd = connection.CreateCommand();
        cmd.CommandText = "SELECT ..."; // your query

        using var reader = cmd.ExecuteReader();

        ...
    }
}
```

Contoh: Koleksi

Koleksi .NET standar tidak aman untuk utas:

```
public class Handler
{
```

```
    private static List<string> items = new List<string>();
    private static Dictionary<string, object> cache = new Dictionary<string, object>();

    public string FunctionHandler(object input, ILambdaContext context)
    {
        items.Add(context.AwsRequestId);
        cache["key"] = input;

        return "Success";
    }
}
```

Gunakan koleksi dari `System.Collections.Concurrent` namespace untuk keamanan konkurensi:

```
public class Handler
{
    private static ConcurrentBag<string> items = new ConcurrentBag<string>();
    private static ConcurrentDictionary<string, object> cache = new
 ConcurrentDictionary<string, object>();

    public string FunctionHandler(object input, ILambdaContext context)
    {
        items.Add(context.AwsRequestId);
        cache["key"] = input;

        return "Success";
    }
}
```

Direktori bersama/tmp

`/tmp`Direktori dibagikan di semua permintaan bersamaan di lingkungan eksekusi. Menulis bersamaan ke file yang sama dapat menyebabkan kerusakan data, misalnya jika permintaan lain menimpa file. Untuk mengatasinya, terapkan penguncian file untuk file bersama atau gunakan nama file unik per permintaan untuk menghindari konflik. Ingatlah untuk membersihkan file yang tidak dibutuhkan untuk menghindari kehabisan ruang yang tersedia.

Pencatatan log

Log interleaving (entri log dari permintaan berbeda yang disisipkan dalam log) adalah normal dalam sistem multi-konkuren. Fungsi yang menggunakan Instans Terkelola Lambda selalu menggunakan

format log JSON terstruktur yang diperkenalkan dengan kontrol logging lanjutan. Format ini mencakup`requestId`, memungkinkan entri log dikorelasikan dengan satu permintaan. Bila Anda menggunakan `context.Logger` objek untuk menghasilkan log, `requestId` secara otomatis disertakan dalam setiap entri log. Untuk informasi lebih lanjut, lihatthe section called “Menggunakan kontrol logging lanjutan Lambda dengan.NET”.

Konteks permintaan

Gunakan `context.AwsRequestId` properti untuk mengakses ID permintaan untuk permintaan saat ini.

Gunakan `context.TraceId` properti untuk mengakses ID X-Ray jejak. Ini memberikan akses aman konkurensi ke ID jejak untuk permintaan saat ini. Lambda tidak mendukung variabel `_X_AMZN_TRACE_ID` lingkungan dengan Instans Terkelola Lambda. ID X-Ray jejak disebarkan secara otomatis saat menggunakan AWS SDK.

Gunakan `ILambdaContext.RemainingTime` untuk mendeteksi batas waktu. Untuk informasi selengkapnya, lihat the section called “Penanganan dan pemulihan kesalahan”.

Contoh: Penanganan batas waktu

Periksa sisa waktu sebelum setiap unit kerja dan hentikan pemrosesan sebelum batas waktu habis. Konfigurasikan buffer berdasarkan durasi yang diharapkan dari potongan pekerjaan Anda berikutnya.

```
private static readonly TimeSpan Buffer = TimeSpan.FromMilliseconds(2000);

public APIGatewayProxyResponse FunctionHandler(APIGatewayProxyRequest request,
 ILambdaContext context)
{
    var items = JsonSerializer.Deserialize<List<string>>(request.Body);
    foreach (var item in items)
    {
        if (context.RemainingTime < Buffer)
            return new APIGatewayProxyResponse { StatusCode = 206, Body = "Timeout
 approaching, stopping early" };
        ProcessItem(item);
    }
    return new APIGatewayProxyResponse { StatusCode = 200, Body = "Done" };
}
```

Contoh: Menyebarkan tenggat waktu ke panggilan hilir

Saat melakukan panggilan ke layanan hilir, sebarkan waktu yang tersisa sebagai batas waktu untuk menghindari gangguan pada panggilan jaringan yang akan hidup lebih lama dari permintaan Anda. Gunakan `CancellationToken` dengan klien bersama daripada membuat klien baru per pemanggilan:

```
using Amazon.S3;

private static readonly IAmazonS3 s3 = new AmazonS3Client();

public async Task<string> FunctionHandler(APIGatewayProxyRequest request,
 ILambdaContext context)
{
    var timeout = context.RemainingTime - TimeSpan.FromMilliseconds(500);
    if (timeout < TimeSpan.FromSeconds(1)) timeout = TimeSpan.FromSeconds(1);

    using var cts = new CancellationTokenSource(timeout);
    await s3.GetObjectAsync("my-bucket", "my-key", cts.Token);
    return "Done";
}
```

Inisialisasi dan shutdown

Inisialisasi fungsi terjadi sekali per lingkungan eksekusi. Objek yang dibuat selama inisialisasi dibagikan di seluruh permintaan.

Untuk fungsi Lambda dengan ekstensi, lingkungan eksekusi memancarkan sinyal SIGTERM saat dimatikan. Sinyal ini digunakan oleh ekstensi untuk memicu tugas pembersihan, seperti pembilasan buffer. Anda dapat berlangganan peristiwa SIGTERM untuk memicu tugas pembersihan fungsi, seperti menutup koneksi database. Untuk mempelajari lebih lanjut tentang siklus hidup lingkungan eksekusi, lihat. the section called “Lingkungan eksekusi”

Versi ketergantungan

Instans Terkelola Lambda memerlukan versi paket minimum berikut:

- Amazon.Lambda.Core: versi 2.7.1 atau yang lebih baru
- Amazon.Lambda.RuntimeSupport: versi 1.14.1 atau yang lebih baru
- OpenTelemetry.Instrumentation.AWSLambda: versi 1.14.0 atau yang lebih baru
- AWSXRayRecorder.Core: versi 2.16.0 atau yang lebih baru
- AWSSDK.Core: versi 4.0.0.32 atau yang lebih baru

Powertools untuk AWS Lambda (.NET)

Powertools untuk AWS Lambda (.NET) AWS dan Distro OpenTelemetry untuk - Instrumentasi DotNet untuk saat ini tidak mendukung Instans Terkelola Lambda.

Langkah selanjutnya

- Tinjau runtime Java untuk Instans Terkelola Lambda
- Tinjau Node.js runtime untuk Instans Terkelola Lambda
- Tinjau runtime Python untuk Instans Terkelola Lambda
- Pelajari tentang penskalaan Instans Terkelola Lambda

Dukungan Rust untuk Instans Terkelola Lambda

Konfigurasi konkurensi

Jumlah maksimum permintaan bersamaan yang dikirim Lambda ke setiap lingkungan eksekusi dikendalikan oleh `PerExecutionEnvironmentMaxConcurrency` pengaturan dalam konfigurasi fungsi. Ini adalah pengaturan opsional, dan nilai default untuk Rust adalah 8 permintaan bersamaan per vCPU, atau Anda dapat mengonfigurasi nilai Anda sendiri. Nilai ini menentukan jumlah tugas Tokio yang dihasilkan oleh runtime dan statis selama masa pakai lingkungan eksekusi. Setiap pekerja menangani tepat satu permintaan dalam penerbangan pada satu waktu, tanpa multiplexing per pekerja. Lambda secara otomatis menyesuaikan jumlah permintaan bersamaan hingga maksimum yang dikonfigurasi berdasarkan kapasitas setiap lingkungan eksekusi untuk menyerap permintaan tersebut.

Membangun fungsi untuk multi-konkurensi

Anda harus menerapkan praktik keamanan thread yang sama saat menggunakan Instans Terkelola Lambda seperti yang Anda lakukan di lingkungan multi-utas lainnya. Karena objek handler dibagikan di semua utas pekerja, setiap status yang dapat berubah harus aman untuk utas. Ini termasuk koleksi, koneksi database, dan objek statis apa pun yang dimodifikasi selama pemrosesan permintaan.

Untuk mengaktifkan penanganan permintaan bersamaan, tambahkan tanda `concurrency-tokio` fitur ke `Cargo.toml` file Anda.

```
[dependencies]
lambda_runtime = { version = "1", features = ["concurrency-tokio"] }
```

T `lambda_runtime::run_concurrent(…)` itik masuk harus dipanggil dari dalam runtime Tokio, biasanya disediakan oleh `#[tokio::main]` atribut pada fungsi utama Anda. Penutupan handler Anda harus menerapkan `Clone` + `Send`. Ini memungkinkan kerangka kerja untuk berbagi handler Anda di beberapa tugas asinkron dengan aman. Jika batas-batas tersebut tidak terpenuhi, kode Anda tidak akan dikompilasi.

Saat Anda membutuhkan status bersama di seluruh pemanggilan (kumpulan database, struct konfigurasi), bungkus dalam `Arc` dan kloning `Arc` ke setiap pemanggilan.

Semua klien AWS SDK untuk Rust aman untuk bersaing dan tidak memerlukan penanganan khusus.

Contoh: AWS Klien SDK

Contoh berikut menggunakan klien S3 untuk mengunggah objek pada setiap pemanggilan. Klien dikloning langsung ke penutupan tanpa: `Arc`

```
let config = aws_config::load_defaults(BehaviorVersion::latest()).await;
let s3_client = aws_sdk_s3::Client::new(&config);

run_concurrent(service_fn(move |event: LambdaEvent<Request>| {
    let s3_client = s3_client.clone(); // cheap clone, no Arc needed
    async move {
        s3_client.put_object()
            .bucket(&event.payload.bucket)
            .key(&event.payload.key)
            .body(event.payload.body.into_bytes().into())
            .send()
            .await?;
        Ok(Response { message: "uploaded".into() })
    }
}))
.await
```

Contoh: Kumpulan koneksi database

Saat handler Anda membutuhkan akses ke status bersama seperti klien dan konfigurasi, bungkus dalam `Arc` dan kloning `Arc` ke setiap pemanggilan:

```
#[derive(Debug)]
struct AppState {
    dynamodb_client: DynamoDbClient,
    table_name: String,
```

```
    cache_ttl: Duration,
}

let config = aws_config::load_defaults(BehaviorVersion::latest()).await;
let state = Arc::new(AppState {
    dynamodb_client: DynamoDbClient::new(&config),
    table_name: std::env::var("TABLE_NAME").expect("TABLE_NAME must be set"),
    cache_ttl: Duration::from_secs(300),
});

run_concurrent(service_fn(move |event: LambdaEvent<Request>| {
    let state = state.clone();
    async move { handle(event, state).await }
}))
.await
```

Direktori /tmp yang dibagikan

Direk `/tmp` tori dibagikan di semua pemanggilan bersamaan di lingkungan eksekusi yang sama. Gunakan nama file unik per pemanggilan (misalnya, sertakan ID permintaan) atau terapkan penguncian file eksplisit untuk menghindari kerusakan data.

Pencatatan

Interleaving log (entri log dari permintaan berbeda yang disisipkan dalam log) normal dalam sistem multi-concurrent. Fungsi yang menggunakan Lambda Managed Instances mendukung format log JSON terstruktur melalui kontrol logging lanjutan Lambda. Format ini mencakup`requestId`, memungkinkan entri log untuk dikorelasikan dengan satu permintaan. Untuk informasi lebih lanjut, lihatthe section called “Menerapkan logging lanjutan dengan peti Tracing”.

Permintaan Konteks

Ob `Context` jek diteruskan langsung ke setiap pemanggilan handler. Gunakan `event.context.request_id` untuk mengakses ID permintaan untuk permintaan saat ini.

Gunakan `event.context.xray_trace_id` untuk mengakses ID X-Ray jejak. Lambda tidak mendukung variabel `_X_AMZN_TRACE_ID` lingkungan dengan Lambda Managed Instances. ID X-Ray jejak disebarkan secara otomatis saat menggunakan AWS SDK untuk Rust.

Gunakan `event.context.deadline` untuk mendeteksi batas waktu - ini berisi tenggat waktu pemanggilan dalam milidetik.

Inisialisasi dan shutdown

Inisialisasi fungsi terjadi sekali per lingkungan eksekusi. Objek yang dibuat selama inisialisasi dibagikan di seluruh permintaan.

Untuk fungsi Lambda dengan ekstensi, lingkungan eksekusi memancarkan sinyal SIGTERM selama shutdown. Sinyal ini digunakan oleh ekstensi untuk memicu tugas pembersihan, seperti buffer pembilasan. `lambda_runtime`menawarkan pembantu untuk menyederhanakan konfigurasi penanganan sinyal shutdown yang anggun, spawn_graceful_shutdown_handler (). Untuk mempelajari lebih lanjut tentang siklus hidup lingkungan eksekusi, lihatthe section called “Lingkungan eksekusi”.

Versi ketergantungan

Instans Terkelola Lambda memerlukan versi paket minimum berikut:

- `lambda_runtime`: versi 1.1.1 atau yang lebih baru, dengan `concurrency-tokio` fitur diaktifkan
- Versi Rust minimum yang didukung (MSRV) adalah 1.84.0.

Jaringan untuk Instans Terkelola Lambda

Saat menjalankan fungsi Instans Terkelola Lambda, Anda perlu mengonfigurasi konektivitas jaringan untuk mengaktifkan Fungsi Peristiwa Lambda untuk mengakses sumber daya di luar VPC. Ini termasuk AWS layanan seperti Amazon S3 dan DynamoDB. Konektivitas juga diperlukan untuk mentransmisikan data telemetri ke CloudWatch Log dan. X-Ray

Opsi konektivitas

Ada tiga pendekatan utama untuk mengonfigurasi konektivitas VPC, masing-masing dengan trade-off yang berbeda untuk biaya, keamanan, dan kompleksitas.

Subnet publik dengan gateway internet

Opsi ini menggunakan subnet publik dengan akses internet langsung melalui gateway internet. Anda dapat memilih antara konfigurasi IPv4 dan IPv6.

IPv4 dengan gateway internet

Untuk mengkonfigurasi konektivitas IPv4 dengan gateway internet

1. Buat atau gunakan subnet publik yang ada dengan blok CIDR IPv4.
2. Lampirkan gateway internet ke VPC Anda.
3. Perbarui tabel rute untuk merutekan `0.0.0.0/0` lalu lintas ke gateway internet.
4. Pastikan sumber daya memiliki alamat IPv4 publik atau alamat IP Elastis yang ditetapkan.
5. Konfigurasikan grup keamanan untuk memungkinkan lalu lintas keluar pada port yang diperlukan.

Konfigurasi ini menyediakan konektivitas dua arah, memungkinkan koneksi keluar dari fungsi Anda dan koneksi masuk dari internet.

IPv6 dengan gateway internet

Untuk mengkonfigurasi konektivitas IPv6 dengan gateway internet

1. Aktifkan IPv6 di VPC Anda.
2. Buat atau gunakan subnet publik yang ada dengan blok CIDR IPv6 yang ditetapkan.
3. Pasang gateway internet ke VPC Anda (gateway internet yang sama dapat menangani IPv4 dan IPv6).
4. Perbarui tabel rute untuk merutekan `::/0` lalu lintas ke gateway internet.
5. Verifikasi bahwa AWS layanan yang Anda perlukan untuk mengakses mendukung IPv6 di Wilayah Anda.
6. Konfigurasikan grup keamanan untuk memungkinkan lalu lintas keluar pada port yang diperlukan.

Konfigurasi ini menyediakan konektivitas dua arah menggunakan pengalamatan IPv6.

IPv6 dengan gateway internet khusus egres

Untuk mengonfigurasi konektivitas IPv6 dengan gateway internet khusus egress

1. Aktifkan IPv6 di VPC Anda.
2. Buat atau gunakan subnet publik yang ada dengan blok CIDR IPv6 yang ditetapkan.
3. Pasang gateway internet khusus egres ke VPC Anda.
4. Perbarui tabel rute untuk merutekan `::/0` lalu lintas ke gateway internet khusus egres.
5. Verifikasi bahwa AWS layanan yang Anda perlukan untuk mengakses mendukung IPv6 di Wilayah Anda.

6. Konfigurasikan grup keamanan untuk memungkinkan lalu lintas keluar pada port yang diperlukan.

Konfigurasi ini menyediakan konektivitas outbound-only, mencegah koneksi masuk dari internet sekaligus memungkinkan fungsi Anda untuk memulai koneksi keluar.

Titik akhir VPC

Titik akhir VPC memungkinkan Anda menghubungkan VPC Anda secara pribadi ke AWS layanan yang didukung tanpa memerlukan gateway internet, perangkat NAT, koneksi VPN, atau koneksi Direct Connect. AWS Lalu lintas antara VPC Anda dan AWS layanan tidak meninggalkan jaringan Amazon.

Untuk mengkonfigurasi titik akhir VPC

1. Buka konsol VPC Amazon di console.aws.amazon. com/vpc/.
2. Di panel navigasi, pilih Titik Akhir.
3. Pilih Buat Titik Akhir.
4. Untuk kategori Layanan, pilih AWS layanan.
5. Untuk nama Layanan, pilih titik akhir layanan yang Anda butuhkan (misalnya, `com.amazonaws.region.s3` untuk Amazon S3).
6. Untuk VPC, pilih VPC Anda.
7. Untuk Subnet, pilih subnet tempat Anda ingin membuat antarmuka jaringan titik akhir. Untuk mendapatkan ketersediaan tinggi, pilih subnet di beberapa Zona Ketersediaan.
8. Untuk grup Keamanan, pilih grup keamanan untuk diasosiasikan dengan antarmuka jaringan titik akhir. Grup keamanan harus mengizinkan lalu lintas masuk dari grup keamanan fungsi Anda pada port yang diperlukan.
9. Pilih Buat titik akhir.

Ulangi langkah-langkah ini untuk setiap AWS layanan yang perlu diakses oleh fungsi Anda.

Subnet pribadi dengan gateway NAT

Opsi ini menggunakan gateway NAT untuk menyediakan akses internet untuk sumber daya di subnet pribadi sambil menjaga sumber daya tetap pribadi.

Untuk mengkonfigurasi subnet pribadi dengan gateway NAT

1. Buat subnet publik (jika belum ada) dengan blok CIDR.
2. Lampirkan gateway internet ke VPC Anda.
3. Buat gateway NAT di subnet publik dan tetapkan alamat IP Elastis.
4. Perbarui tabel rute subnet publik untuk menambahkan rute: `0.0.0.0/0` → gateway internet.
5. Buat atau gunakan subnet pribadi yang ada dengan blok CIDR.
6. Perbarui tabel rute subnet pribadi untuk menambahkan rute: `0.0.0.0/0` → NAT gateway.
7. Konfigurasikan grup keamanan untuk memungkinkan lalu lintas keluar pada port yang diperlukan.

Untuk ketersediaan tinggi, gunakan satu gateway NAT di setiap Availability Zone dan konfigurasikan tabel rute per Availability Zone untuk menggunakan gateway NAT lokal. Ini mencegah biaya transfer data lintas-AZ dan meningkatkan ketahanan.

Memilih opsi konektivitas

Pertimbangkan faktor-faktor berikut saat memilih opsi konektivitas:

Subnet publik dengan gateway internet

- Konfigurasi paling sederhana dengan biaya terendah
- Cocokkan untuk lingkungan pengembangan dan pengujian
- Sumber daya dapat menerima koneksi masuk dari internet (pertimbangan keamanan)
- Mendukung IPv4 dan IPv6

Titik akhir VPC

- Keamanan tertinggi, lalu lintas tetap dalam AWS jaringan
- Latensi lebih rendah dibandingkan dengan perutean internet
- Direkomendasikan untuk lingkungan produksi dengan persyaratan keamanan yang ketat
- Biaya per titik akhir yang lebih tinggi, per Availability Zone, dan per GB yang diproses
- Memerlukan titik akhir di setiap Availability Zone untuk ketersediaan tinggi

Subnet pribadi dengan gateway NAT

- Sumber daya tetap pribadi tanpa akses internet masuk

- Pola arsitektur perusahaan standar
- Mendukung semua lalu lintas internet IPv4
- Biaya moderat dengan gateway NAT per jam dan biaya pemrosesan data
- Hanya mendukung IPv4

Langkah selanjutnya

- Pelajari tentang penyedia kapasitas untuk Instans Terkelola Lambda
- Memahami penskalaan untuk Instans Terkelola Lambda
- Tinjau panduan khusus runtime untuk Java, Node.js, dan Python
- Memahami keamanan dan izin untuk Instans Terkelola Lambda

Memantau Instans Terkelola Lambda

Lambda menyediakan dua jenis data pemantauan untuk penyedia kapasitas Anda: metrik dan log sistem. Bersama-sama, ini memberi Anda visibilitas ke dalam pemanfaatan sumber daya, peristiwa siklus hidup instans, dan masalah operasional.

- Metrik — Lambda secara otomatis menerbitkan CloudWatch metrik untuk penyedia kapasitas Anda untuk membantu Anda memantau pemanfaatan sumber daya, melacak biaya, dan mengoptimalkan kinerja. Untuk informasi selengkapnya, lihat CloudWatch metrik untuk Instans Terkelola Lambda.
- Log sistem — Lambda secara otomatis mengirimkan log sistem ke Log CloudWatch yang merekam peristiwa siklus hidup instans dan kesalahan untuk penyedia kapasitas Anda. Lihat informasi yang lebih lengkap di Log sistem penyedia kapasitas untuk Instans Terkelola Lambda.

CloudWatch metrik untuk Instans Terkelola Lambda

Anda dapat memantau Instans Terkelola Lambda menggunakan CloudWatch metrik. Lambda secara otomatis menerbitkan metrik CloudWatch untuk membantu Anda memantau pemanfaatan sumber daya, melacak biaya, dan mengoptimalkan kinerja.

Metrik yang tersedia

Instans Terkelola Lambda menyediakan metrik pada dua tingkat: tingkat penyedia kapasitas dan tingkat lingkungan eksekusi.

Metrik tingkat penyedia kapasitas

Metrik tingkat penyedia kapasitas memberikan visibilitas ke seluruh pemanfaatan sumber daya di seluruh instans Anda. Metrik ini menggunakan dimensi berikut:

- CapacityProviderName- Nama penyedia kapasitas Anda
- InstanceType- Jenis instance EC2

Metrik pemanfaatan sumber daya berikut tersedia:

- CPUUtilization - Persentase pemanfaatan CPU di seluruh instance di penyedia kapasitas
- MemoryUtilization- Persentase pemanfaatan memori di seluruh instance di penyedia kapasitas

Metrik kapasitas berikut tersedia:

- vcpuAvailable - Jumlah vCPU yang tersedia pada instans untuk alokasi (dalam hitungan)
- MemoryAvailable- Jumlah memori yang tersedia pada instance untuk alokasi (dalam byte)
- vCPU - Jumlah vCPU yang dialokasikan pada instance untuk lingkungan eksekusi (dalam hitungan)
- MemoryAllocated- Jumlah memori yang dialokasikan pada instance untuk lingkungan eksekusi (dalam byte)

Metrik tingkat lingkungan eksekusi

Metrik tingkat lingkungan eksekusi memberikan visibilitas ke dalam pemanfaatan sumber daya dan konkurensi untuk fungsi individu. Metrik ini menggunakan dimensi berikut:

- CapacityProviderName- Nama penyedia kapasitas Anda
- FunctionName- Nama fungsi Lambda Anda
- Sumber Daya - Berdasarkan sumber daya, lihat metrik untuk versi fungsi tertentu.

> **Note**
>
> Untuk Lambda Managed Instances (LMI), `Resource` dimensi hanya mendukung versi fungsi. Formatnya adalah `<FunctionName>:<FunctionVersion>`.

Metrik lingkungan eksekusi yang tersedia:

- ExecutionEnvironmentConcurrency- Konkurensi maksimum selama periode sampel 1 menit
- ExecutionEnvironmentConcurrencyLimit- Batas konkurensi maksimum per lingkungan eksekusi
- ExecutionEnvironmentCPUUtilization- Persentase pemanfaatan CPU untuk lingkungan eksekusi fungsi
- ExecutionEnvironmentMemoryUtilization- Persentase pemanfaatan memori untuk lingkungan eksekusi fungsi

Metrik alasan throttle

Instans Terkelola Lambda memancarkan metrik alasan throttle granular yang mengidentifikasi batasan sumber daya yang menyebabkan throttle. Untuk setiap throttle, Lambda memancarkan tepat salah satu sub-metrik berikut dengan nilai 1, sedangkan tiga sisanya dipancarkan dengan nilai 0:

- ConcurrencyThrottles— Lingkungan eksekusi mencapai batas konkurensi maksimumnya. Ini dapat diatasi dengan meningkatkan ExecutionEnvironmentMaxConcurrency atau dengan men skalakan lingkungan eksekusi secara lebih agresif.
- CPUThrottles — Lingkungan eksekusi menghabiskan sumber daya CPU yang dialokasikan. Hal ini dapat diatasi dengan meningkatkan alokasi vCPU fungsi Anda atau dengan mengurangi beban per lingkungan ExecutionEnvironmentMaxConcurrency yang lebih rendah.
- MemoryThrottles- Lingkungan eksekusi menghabiskan memori yang dialokasikan. Hal ini dapat diatasi dengan meningkatkan alokasi memori fungsi Anda atau dengan mengurangi beban per lingkungan ExecutionEnvironmentMaxConcurrency menjadi lebih rendah.
- DiskThrottles- Lingkungan eksekusi menghabiskan ruang disk yang dialokasikan. Hal ini dapat diatasi dengan meningkatkan penyimpanan sementara fungsi Anda atau dengan mengurangi beban per lingkungan ExecutionEnvironmentMaxConcurrency yang lebih rendah.

Lambda selalu memancarkan `Throttles` metrik standar di samping sub-metrik ini. Gunakan metrik ini untuk memahami mengapa fungsi Anda melakukan penskalaan dan apakah akan menyesuaikan konkurensi maksimum, memori fungsi, alokasi vCPU, atau konfigurasi disk. Untuk daftar lengkap metrik Lambda, lihatJenis metrik untuk Fungsi Peristiwa Lambda.

Frekuensi metrik dan retensi

Metrik Instans Terkelola Lambda dipublikasikan dengan interval 1 menit dan disimpan selama 15 bulan.

Melihat metrik di CloudWatch

Untuk melihat metrik Instans Terkelola Lambda di konsol CloudWatch

1. Buka CloudWatch konsol di console.aws.amazon. com/cloudwatch/.
2. Pada panel navigasi, silakan pilih Metrik.
3. Di tab Semua metrik, pilih AWS/Lambda.
4. Pilih dimensi metrik yang ingin Anda lihat:
   - Untuk metrik tingkat penyedia kapasitas, filter berdasarkan CapacityProviderName dan InstanceType
   - Untuk metrik tingkat lingkungan eksekusi, filter berdasarkan CapacityProviderName, FunctionName, dan Sumber Daya
5. Pilih metrik yang ingin Anda pantau.

Menggunakan metrik untuk mengoptimalkan kinerja

Pantau pemanfaatan CPU dan memori untuk memahami apakah fungsi Anda berukuran tepat. Pemanfaatan tinggi mungkin menunjukkan perlunya jenis instance yang lebih besar atau peningkatan alokasi memori fungsi. Lacak metrik konkurensi untuk memahami perilaku penskalaan dan mengidentifikasi potensi pelambatan.

Pantau metrik kapasitas untuk memverifikasi sumber daya yang cukup tersedia untuk beban kerja Anda. VCPUAvailable dan MemoryAvailable metrik membantu Anda memahami kapasitas yang tersisa pada instans Anda.

Langkah selanjutnya

- Pelajari tentang penskalaan Instans Terkelola Lambda
- Tinjau panduan khusus runtime untuk Java, Node.js, dan Python ???
- Konfigurasikan konektivitas VPC untuk penyedia kapasitas Anda
- Memahami keamanan dan izin untuk Instans Terkelola Lambda

Log sistem penyedia kapasitas untuk Instans Terkelola Lambda

Untuk membantu Anda memantau dan memecahkan masalah lingkungan Instans Terkelola Lambda, Lambda secara otomatis membuat log sistem untuk penyedia kapasitas Anda. Anda dapat melihat log untuk penyedia kapasitas menggunakan konsol Lambda, CloudWatch konsol, AWS Command

Line Interface (AWS CLI), atau CloudWatch API. Anda juga dapat merutekan log ke Amazon S3 dan Firehose menggunakan filter langganan CloudWatch Logs.

Selama peran operator penyedia kapasitas Anda memiliki izin yang diperlukan, Lambda menangkap log sistem dan mengirimkannya ke Amazon CloudWatch Logs, yang merupakan tujuan default.

> **Note**
>
> Log sistem penyedia kapasitas diaktifkan secara default. Lambda mengirimkan log sistem ke CloudWatch Log secara otomatis saat Anda membuat penyedia kapasitas.

- CloudWatch Log adalah tujuan pencatatan default untuk log sistem penyedia kapasitas. CloudWatch Log menyediakan kemampuan tampilan dan analisis log real-time, dengan dukungan untuk membuat metrik dan alarm berdasarkan data log Anda.
- Amazon S3 ekonom is untuk penyimpanan jangka panjang, dan layanan seperti Athena dapat digunakan untuk menganalisis log. Latensi biasanya lebih tinggi.
- Firehose menawarkan streaming log terkelola ke berbagai tujuan. Jika Anda perlu mengirim log ke AWS layanan lain (misalnya, OpenSearch Service atau Redshift Data API) atau platform pihak ketiga (seperti Datadog, New Relic, atau Splunk), Firehose menyederhanakan proses itu dengan menyediakan integrasi pra-build.

Mengkonfigurasi tujuan log

Lambda mendukung beberapa tujuan untuk log sistem penyedia kapasitas Anda. Terlepas dari tujuan yang Anda pilih, Lambda menyediakan opsi untuk mengontrol penyaringan log dan pengiriman.

Log sistem penyedia kapasitas menggunakan format terstruktur JSON secara eksklusif. Log terstruktur JSON memberikan peningkatan kemampuan pencarian dan memungkinkan analisis otomatis. Anda dapat mengontrol log mana yang dikirim Lambda ke tujuan pilihan Anda dengan mengonfigurasi tingkat log sistem. Pemfilteran membantu Anda mengelola biaya penyimpanan dan membuatnya lebih mudah untuk menemukan entri log yang relevan selama debugging.

Untuk informasi selengkapnya tentang menyiapkan setiap tujuan, lihat topik berikut:

- Mengirim log sistem penyedia kapasitas ke CloudWatch Log
- Mengirim log sistem penyedia kapasitas ke Firehose
- Mengirim log sistem penyedia kapasitas ke Amazon S3

Mengkonfigurasi kontrol logging untuk penyedia kapasitas

Untuk memberi Anda kontrol lebih besar atas bagaimana log sistem penyedia kapasitas Anda ditangkap, diproses, dan dikonsumsi, Lambda menawarkan opsi konfigurasi logging berikut:

- Tingkat log — pilih tingkat detail log sistem yang dikirim Lambda ke CloudWatch Log:`DEBUG`,`INFO`, atau`WARN`.
- Grup log — pilih grup CloudWatch log log yang dikirim oleh penyedia kapasitas Anda log.

Untuk informasi selengkapnya tentang mengonfigurasi kontrol logging, lihat topik berikut:

- Log-level penyaringan untuk log sistem penyedia kapasitas
- Format log sistem penyedia kapasitas
- Referensi peristiwa log sistem penyedia kapasitas

Format log sistem penyedia kapasitas

Log sistem penyedia kapasitas menggunakan format JSON terstruktur. Log terstruktur JSON memberikan peningkatan kemampuan pencarian dan memungkinkan analisis otomatis peristiwa operasional penyedia kapasitas Anda.

Format JSON untuk log sistem

Setiap entri log sistem ditangkap sebagai objek JSON yang berisi pasangan kunci-nilai dengan bidang tingkat atas berikut:

Bidang	Tipe	Deskripsi
`timestamp`	String	Waktu peristiwa log dihasilkan, dalam format ISO 8601.
`level`	String	Tingkat log acara.
`message`	String	Deskripsi peristiwa yang dapat dibaca manusia.
`metadata`	Objek	Metadata terstruktur tentang acara tersebut. Berisi bidang yang bervariasi menurut jenis acara.

Kolom metadata

Ob `metadata` jek berisi bidang berikut. Tidak semua bidang hadir di setiap acara.

Bidang	Tipe	Deskripsi
`instanceId`	String	ID instans Amazon EC2 yang terkait dengan acara.
`availabilityZones`	Array string	Zona Ketersediaan yang terkait dengan acara.
`instanceTypes`	Array string	Jenis instans Amazon EC2 yang terkait dengan acara tersebut.
`capacityProviderArn`	String	ARN dari penyedia kapasitas.

Contoh entri log

Contoh berikut menunjukkan entri log sistem untuk peluncuran instance yang berhasil:

```
{
  "timestamp": "2025-07-01T14:31:15.456Z",
  "level": "INFO",
  "message": "Successfully launched c7g.xlarge instance i-0abc123def456789 in us-
east-1a",
  "metadata": {
    "instanceId": "i-0abc123def456789",
    "availabilityZones": ["us-east-1a"],
    "instanceTypes": ["c7g.xlarge"],
    "capacityProviderArn": "arn:aws:lambda:us-east-1:123456789012:capacity-provider:my-
capacity-provider"
  }
}
```

Log-level penyaringan untuk log sistem penyedia kapasitas

Lambda dapat memfilter log sistem penyedia kapasitas Anda sehingga hanya log tingkat detail tertentu atau lebih rendah yang dikirim ke CloudWatch Log. Anda dapat memilih antara level log berikut.

Tingkat log	Penggunaan
DEBUG (paling detail)	Informasi terperinci untuk debugging sistem
INFO	Pesan yang merekam operasi normal penyedia kapasitas
PERINGATAN (paling sedikit detail)	Pesan tentang potensi kesalahan yang dapat menyebabkan perilaku tak terduga jika tidak ditangani

Ketika Anda memilih tingkat log, Lambda mengirimkan log pada level itu dan lebih rendah. Misalnya, jika Anda menyetel level log sistem penyedia kapasitas ke INFO, Lambda tidak mengirim output log di tingkat DEBUG.

Secara default, Lambda mengatur level log sistem ke INFO. Untuk menerima log yang lebih rinci, atur level ke DEBUG. Untuk menerima lebih sedikit log, atur level ke WARN. Untuk melihat daftar level log yang dipetakan Lambda ke berbagai peristiwa log sistem, lihatReferensi peristiwa log sistem penyedia kapasitas.

Mengkonfigurasi tingkat log sistem

Untuk mengonfigurasi level log sistem untuk penyedia kapasitas Anda, Anda dapat menggunakan konsol Lambda atau AWS Command Line Interface (AWS CLI). Anda juga dapat mengkonfigurasi level log sistem menggunakan perintah `CreateCapacityProvider` dan `UpdateCapacityProvider` Lambda API.

Untuk mengkonfigurasi level log sistem (konsol)

1. Buka halaman Penyedia kapasitas pada konsol Lambda.
2. Pilih penyedia kapasitas.
3. Pilih Konfigurasi tambahan.
4. Di bagian Konfigurasi logging, pilih Edit.
5. Pilih tingkat log sistem yang Anda inginkan untuk penyedia kapasitas Anda.

6. Pilih Simpan.

Untuk mengatur tingkat log sistem saat membuat penyedia kapasitas (AWS CLI)

- Untuk mengonfigurasi tingkat log sistem saat Anda membuat penyedia kapasitas baru, gunakan `--telemetry-config` opsi dalam perintah create-capac ity-provider. Set `SystemLogLevel` el ke salah satu`DEBUG`,`INFO`, atau`WARN`.

```
aws lambda create-capacity-provider \
  --capacity-provider-name my-capacity-provider \
  --vpc-config SubnetIds=subnet-12345,subnet-67890,SecurityGroupIds=sg-12345 \
  --permissions-config
 CapacityProviderOperatorRoleArn=arn:aws:iam::123456789012:role/MyOperatorRole \
  --telemetry-config LoggingConfig={SystemLogLevel=DEBUG}
```

Untuk memperbarui tingkat log sistem dari penyedia kapasitas yang ada (AWS CLI)

- Untuk mengubah tingkat log sistem penyedia kapasitas yang ada, gunakan perintah update-capac ity-provider.

```
aws lambda update-capacity-provider \
  --capacity-provider-name my-capacity-provider \
  --telemetry-config LoggingConfig={SystemLogLevel=WARN}
```

Referensi peristiwa log sistem penyedia kapasitas

Lambda menerbitkan peristiwa log sistem untuk operasi siklus hidup instance penyedia kapasitas. Tabel berikut menunjukkan peristiwa yang diterbitkan Lambda dan apakah setiap peristiwa dicatat berdasarkan tingkat log sistem yang dikonfigurasi.

Untuk informasi selengkapnya tentang mengkonfigurasi level log sistem, lihatLog-level penyaringan untuk log sistem penyedia kapasitas. Untuk detail tentang format entri log, lihatFormat log sistem penyedia kapasitas.

Pemetaan tingkat peristiwa log sistem

Peristiwa	Deskripsi	**DEBUG**	**INFO**	**WARN**
Peluncuran	Dipancarkan saat Lambda memulai peluncuran instance untuk penyedia kapasitas.	Tercatat	Tercatat	Tidak tercatat
Diluncurkan	Dipancarkan ketika instance berhasil bertransisi ke keadaan aktif dan siap melayani pemanggilan.	Tercatat	Tercatat	Tidak tercatat
LaunchFailed	Dipancarkan saat upaya peluncuran instance gagal.	Tercatat	Tercatat	Tercatat
Mengakhiri	Dipancarkan saat Lambda menandai instance untuk penghentian.	Tercatat	Tercatat	Tidak tercatat
Diakhiri	Dipancarkan saat instance berhasil dihentikan.	Tercatat	Tercatat	Tidak tercatat
TerminateFailed	Dipancarkan saat Lambda gagal menghentikan instance.	Tercatat	Tercatat	Tercatat
HealthCheckFailed	Dipancarkan ketika instance gagal dalam pemeriksaan kesehatan dan ditandai sebagai tidak sehat.	Tercatat	Tercatat	Tercatat

> **Note**
>
> Daftar ini diperluas saat Lambda menambahkan kemampuan penyedia kapasitas baru.

Mengirim log sistem penyedia kapasitas ke CloudWatch Log

Secara default, Lambda secara otomatis menangkap log sistem untuk penyedia kapasitas Anda dan mengirimkannya ke CloudWatch Log, asalkan peran operator penyedia kapasitas Anda memiliki izin yang diperlukan. Log ini disimpan dalam grup log bernama`/aws/lambda/capacity-provider/<capacity-provider-name>`. Jika diperlukan, Anda dapat mengonfigurasi penyedia kapasitas untuk mengirim log ke grup log yang berbeda menggunakan konsol Lambda, AWS CLI, atau Lambda API. Untuk informasi selengkapnya, lihat Mengkonfigurasi CloudWatch grup log.

Anda dapat melihat log untuk penyedia kapasitas menggunakan konsol Lambda, CloudWatch konsol, AWS Command Line Interface (AWS CLI), atau CloudWatch API. Untuk informasi selengkapnya, lihat Meli CloudWatch hat log untuk penyedia kapasitas.

Izin IAM yang diperlukan

Peran operator penyedia kapasitas Anda memerlukan izin berikut untuk mengunggah CloudWatch log ke Log:

- `logs:CreateLogGroup`
- `logs:CreateLogStream`
- `logs:PutLogEvents`

Untuk mempelajari selengkapnya, lihat Menggunakan kebijakan berbasis identitas (kebijakan IAM) untuk CloudWatch Log di Panduan Pengguna Amazon CloudWatch .

Anda dapat menambahkan izin CloudWatch Log ini menggunakan kebijakan terkel `AWSLambdaManagedEC2ResourceOperator` AWS ola yang disediakan oleh Lambda. Untuk menambahkan kebijakan ini ke peran Anda, jalankan perintah berikut:

```
aws iam attach-role-policy --role-name your-role --policy-arn arn:aws:iam::aws:policy/
AWSLambdaManagedEC2ResourceOperator
```

Untuk informasi selengkapnya, lihat the section called “Peran operator”.

> **Important**
>
> Jika peran operator tidak memiliki izin yang diperlukan, Lambda tidak mengirimkan log sistem dan tidak mengembalikan kesalahan. Pastikan peran operator Anda memiliki izin yang benar untuk menghindari kegagalan pengiriman log secara diam-diam.

Harga

Tidak ada biaya tambahan untuk menggunakan log sistem penyedia kapasitas; namun, biaya CloudWatch Log standar berlaku. Untuk informasi lebih lanjut, lihat Harga CloudWatch .

Mengkonfigurasi CloudWatch grup log

Secara default, Lambda membuat grup log bernama `/aws/lambda/capacity-provider/<capacity-provider-name>` untuk penyedia kapasitas Anda. Untuk mengirim log ke grup log yang berbeda, gunakan konsol Lambda AWS CLI,, atau perintah `UpdateCapacityProvider` API. `CreateCapacityProvider`

Saat Anda mengonfigurasi grup log kustom, nama harus mengikuti aturan penamaan CloudWatch Log. Selain itu, nama grup log kustom tidak boleh dimulai dengan string`aws/`. Jika Anda membuat grup log kustom dimulai dengan`aws/`, Lambda tidak dapat membuat grup log, dan penyedia kapasitas Anda tidak mengirim log ke CloudWatch.

Pengaliran Log

Dalam grup log, Lambda membuat aliran log menggunakan format penamaan berikut:

```
<capacity-provider-name>/managed-instances
```

Misalnya, untuk penyedia kapasitas bernama`my-capacity-provider`, aliran log adalah:

```
my-capacity-provider/managed-instances
```

Mengkonfigurasi grup log kustom

Untuk mengubah grup log penyedia kapasitas (konsol)

1. Buka halaman Penyedia kapasitas pada konsol Lambda.
2. Pilih penyedia kapasitas.
3. Pilih Konfigurasi tambahan.
4. Di bagian Konfigurasi logging, pilih Edit.
5. Masukkan nama grup CloudWatch log log tempat penyedia kapasitas Anda mengirim log. Jika Anda memasukkan nama grup log yang ada, penyedia kapasitas Anda menggunakan grup itu. Jika tidak ada grup log dengan nama yang Anda masukkan, Lambda membuat grup log baru dengan nama itu.

6. Pilih Simpan.

Untuk mengubah grup log penyedia kapasitas (AWS CLI)

- Untuk mengubah grup log dari penyedia kapasitas yang ada, gunakan perintah update-capacity-provider.

```
aws lambda update-capacity-provider \
  --capacity-provider-name my-capacity-provider \
  --telemetry-config LoggingConfig={LogGroup=myLogGroup}
```

Untuk menentukan grup log kustom saat Anda membuat penyedia kapasitas (AWS CLI)

- Untuk menentukan grup log kustom saat Anda membuat penyedia kapasitas baru, gunakan `--telemetry-config` opsi dalam perintah create-capacity-provider.

```
aws lambda create-capacity-provider \
  --capacity-provider-name my-capacity-provider \
  --vpc-config SubnetIds=subnet-12345,subnet-67890,SecurityGroupIds=sg-12345 \
  --permissions-config
 CapacityProviderOperatorRoleArn=arn:aws:iam::123456789012:role/MyOperatorRole \
  --telemetry-config LoggingConfig={LogGroup=myLogGroup}
```

Izin peran operator

Agar penyedia kapasitas Anda mengirim CloudWatch log ke Log, peran operator harus memiliki PutLogEvents izin log:. Lambda tidak menambahkan izin ini secara otomatis. Jika peran operator belum memilikinya, tambahkan sebelum Anda membuat penyedia kapasitas. Izin ini disertakan dalam kebijakan ter `AWSLambdaManagedEC2ResourceOperator` kelola.

Untuk informasi selengkapnya tentang peran operator, lihatthe section called “Peran operator”.

Meli CloudWatch hat log untuk penyedia kapasitas

Anda dapat melihat CloudWatch log Amazon Logs untuk penyedia kapasitas Anda menggunakan konsol Lambda, CloudWatch konsol, atau AWS Command Line Interface (AWS CLI). Ikuti petunjuk di bagian berikut untuk mengakses log penyedia kapasitas Anda.

Streaming log dengan CloudWatch Logs Live Tail

Amazon CloudWatch Logs Live Tail membantu Anda memecahkan masalah penyedia kapasitas dengan cepat dengan menampilkan daftar streaming peristiwa log baru secara real time. Anda dapat melihat dan memfilter log yang dicerna dari penyedia kapasitas Anda, membantu Anda mendeteksi dan menyelesaikan masalah dengan cepat.

> **Note**
>
> Sesi Live Tail dikenakan biaya berdasarkan waktu penggunaan sesi, per menit. Untuk informasi selengkapnya tentang harga, lihat CloudWatch Harga Amazon.

Untuk memulai sesi Live Tail, buka grup log penyedia kapasitas Anda di CloudWatch konsol dan pilih Live Tail. Untuk informasi selengkapnya, lihat Menggunakan Live Tail untuk melihat log hampir secara real time di Panduan CloudWatch Pengguna Amazon.

Lihat log di konsol Lambda

Untuk melihat log di konsol Lambda

1. Buka halaman Penyedia kapasitas pada konsol Lambda.
2. Pilih penyedia kapasitas.
3. Pilih tab Pantau.
4. Pilih Lihat log in CloudWatch untuk membuka grup log di CloudWatch konsol.

Lihat log di CloudWatch konsol

Untuk melihat log di CloudWatch konsol

1. Buka CloudWatch konsol di console.aws.amazon. com/cloudwatch/.
2. Di panel navigasi, pilih Log, lalu Grup log.
3. Pilih grup log untuk penyedia kapasitas Anda. Secara default, grup log diberi nama `/aws/lambda/capacity-provider/<capacity-provider-name>`.
4. Pilih aliran log untuk melihat peristiwa log.

Lihat log dengan AWS CLI

Untuk melihat log menggunakan AWS CLI, gunakan `aws logs` perintah. Contoh berikut mengambil peristiwa log terbaru untuk penyedia kapasitas:

```
aws logs get-log-events \
  --log-group-name /aws/lambda/capacity-provider/my-capacity-provider \
  --log-stream-name my-capacity-provider/managed-instances
```

Anda juga dapat menggunakan CloudWatch Logs Insights untuk menanyakan dan menganalisis log sistem penyedia kapasitas Anda. Contoh kueri berikut untuk semua peristiwa tingkat ERROR:

```
aws logs start-query \
  --log-group-name /aws/lambda/capacity-provider/my-capacity-provider \
  --start-time $(date -d '1 hour ago' +%s) \
  --end-time $(date +%s) \
  --query-string 'fields @timestamp, @message | filter level = "ERROR" | sort
 @timestamp desc'
```

Mengirim log sistem penyedia kapasitas ke Firehose

Anda dapat mengirim log sistem penyedia kapasitas ke Amazon Data Firehose. Dengan Firehose, Anda dapat melakukan streaming log secara real time ke berbagai tujuan, termasuk alat analisis pihak ketiga dan titik akhir khusus.

> **Note**
>
> Anda dapat mengonfigurasi log sistem penyedia kapasitas untuk dikirim ke Firehose menggunakan AWS CLI, AWS CloudFormation, dan semua AWS SDK.

Harga

Untuk informasi selengkapnya tentang harga, lihat CloudWatch Harga Amazon.

Menyiapkan pengiriman log Firehose

Untuk mengirim log sistem penyedia kapasitas ke Firehose, buat filter langganan CloudWatch Log pada grup log penyedia kapasitas Anda. Filter langganan merutekan peristiwa log ke aliran pengiriman Firehose Anda.

Secara default, grup log penyedia kapasitas Anda diberi nama`/aws/lambda/capacity-provider/<capacity-provider-name>`. Gunakan grup log ini saat membuat filter langganan.

Untuk petunjuk tentang cara menyiapkan filter langganan untuk Firehose, termasuk pembuatan peran IAM dan izin, lihat Filter berlangganan dengan Firehose di Panduan Pengguna Amazon CloudWatch .

Cross-account penebangan

Anda dapat mengonfigurasi penyedia kapasitas untuk mengirim log ke aliran pengiriman Firehose di AWS akun lain. Ini memerlukan pengaturan tujuan dan konfigurasi izin yang sesuai di kedua akun.

Untuk petunjuk tentang cara menyiapkan log lintas akun, termasuk peran dan kebijakan IAM yang diperlukan, lihat Men yiapkan langganan lintas-akun baru di Panduan Pengguna Amazon CloudWatch .

Mengirim log sistem penyedia kapasitas ke Amazon S3

Anda dapat mengirim log sistem penyedia kapasitas ke Amazon Simple Storage Service. Dengan Amazon S3, Anda dapat menyimpan log secara hemat biaya untuk penyimpanan jangka panjang dan menganalisisnya menggunakan layanan seperti Athena.

> **Note**
>
> Anda dapat mengonfigurasi log sistem penyedia kapasitas untuk dikirim ke Amazon S3 menggunakan AWS CLI, AWS CloudFormation, dan semua AWS SDK.

Harga

Untuk informasi selengkapnya tentang harga, lihat CloudWatch Harga Amazon.

Menyiapkan pengiriman log Amazon S3

Untuk mengirim log sistem penyedia kapasitas ke Amazon S3, buat filter langganan CloudWatch Log pada grup log penyedia kapasitas Anda. Filter langganan merutekan peristiwa log ke bucket Amazon S3 Anda.

Secara default, grup log penyedia kapasitas Anda diberi nama`/aws/lambda/capacity-provider/<capacity-provider-name>`. Gunakan grup log ini saat membuat filter langganan.

Untuk petunjuk tentang cara menyiapkan filter langganan untuk Amazon S3, termasuk pembuatan peran IAM dan izin bucket, lihatMengirim log Fungsi Peristiwa Lambda ke Amazon S3.

Cross-account penebangan

Anda dapat mengonfigurasi penyedia kapasitas untuk mengirim log ke bucket Amazon S3 di AWS akun lain. Ini memerlukan pengaturan tujuan dan konfigurasi izin yang sesuai di kedua akun.

Untuk petunjuk tentang cara menyiapkan log lintas akun, termasuk peran dan kebijakan IAM yang diperlukan, lihat Men yiapkan langganan lintas-akun baru di Panduan Pengguna Amazon CloudWatch .

Kuota Instans Terkelola Lambda

Halaman ini menjelaskan kuota layanan untuk Instans Terkelola AWS Lambda. Kuota ini terpisah dari kuota AWS Lambda (default). Beberapa kuota dapat ditingkatkan berdasarkan permintaan.

Selain kuota yang tercantum di halaman ini, Instans Terkelola Lambda tunduk pada kuota layanan Amazon EC2 yang diterapkan ke akun Anda. Kuota yang biasanya memengaruhi pelanggan Instans Terkelola Lambda mencakup batas vCPU tipe instans EC2, batas volume Amazon EBS, dan alamat IP yang tersedia di subnet VPC Anda.

Kuota permintaan API Lambda

Kuota ini mengontrol tingkat di mana Anda dapat melakukan panggilan API untuk mengelola penyedia kapasitas Instans Terkelola Lambda. Batas tarif API baca dan tulis berlaku untuk semua operasi penyedia kapasitas yang digabungkan, termasuk membuat, memperbarui, mendeskripsikan, dan menghapus penyedia kapasitas.

Sumber daya	Kuota
Tingkat gabungan maksimum (permintaan per detik) untuk semua penyedia kapasitas membaca API	15 permintaan per detik. Tidak dapat ditingkatkan.
Tingkat gabungan maksimum (permintaan per detik) untuk semua API penulisan penyedia kapasitas	1 permintaan per detik. Tidak dapat ditingkatkan.

Kuota sumber daya Instans Terkelola Lambda

Kuota ini menentukan batas sumber daya Instance Terkelola Lambda inti dalam akun Anda. AWS Mereka mengatur jumlah penyedia kapasitas yang dapat Anda buat dan jumlah versi fungsi yang dapat dikaitkan dengan masing-masing penyedia kapasitas.

Sumber daya	Kuota
Penyedia kapasitas	1.000. Jumlah maksimum penyedia kapasitas yang dibuat di akun.
Versi fungsi per penyedia kapasitas	100. Jumlah maksimum versi fungsi per penyedia kapasitas. Tidak dapat ditingkatkan.
vCPU per penyedia kapasitas	15.000. Jumlah maksimum vCPU per penyedia kapasitas.

Kuota pemetaan sumber acara

Kuota ini mengontrol batas throughput dan konfigurasi untuk memproses peristiwa dari berbagai AWS layanan di Instans Terkelola Lambda. Batas throughput memastikan kinerja yang dapat diprediksi sementara batas jumlah pemetaan membantu menjaga stabilitas layanan. Pemetaan sumber peristiwa di Instans Terkelola Lambda mendukung Amazon SQS, DynamoDB Streams, Amazon Kinesis Data Streams, Amazon MSK, dan Apache Kafka yang dikelola sendiri sebagai sumber acara.

Sumber daya	Kuota
Throughput pemetaan sumber peristiwa SQS standar pada Instans Terkelola Lambda	5 MB per detik. Tidak dapat ditingkatkan.
Throughput pemetaan sumber acara Kafka standar pada Instans Terkelola Lambda	1 MB per detik. Tidak dapat ditingkatkan.
Pemetaan sumber acara Kafka standar pada Instans Terkelola Lambda	100 pemetaan sumber acara. Tidak dapat ditingkatkan.

Sumber daya	Kuota
Throughput pemetaan sumber peristiwa Kinesis pada Instans Terkelola Lambda	25 MB per detik. Dapat ditingkatkan.
Throughput pemetaan sumber peristiwa DynamoDB pada Instans Terkelola Lambda	10 MB per detik. Dapat ditingkatkan.
Memanggil throughput permintaan untuk pemanggilan asinkron pada Instans Terkelola Lambda	5 MB per detik. Dapat ditingkat kan.

Meminta peningkatan kuota

Untuk kuota yang dapat ditingkatkan, Anda dapat meminta peningkatan melalui konsol Service Quotas.

Untuk meminta kenaikan kuota

1. Buka konsol Service Quotas di console.aws.amazon. com/servicequotas/.
2. Di panel navigasi, pilih Layanan AWS .
3. Pilih AWS Lambda.
4. Pilih kuota yang ingin Anda tingkatkan.
5. Pilih Ajukan peningkatan kuota.
6. Masukkan nilai kuota baru dan berikan pembenaran untuk kenaikan tersebut.
7. Pilih Minta.

Langkah selanjutnya

- Pelajari tentang penyedia kapasitas untuk Instans Terkelola Lambda
- Memahami penskalaan untuk Instans Terkelola Lambda
- Tinjau panduan khusus runtime untuk Java, Node.js, dan Python
- Konfigurasikan konektivitas VPC untuk penyedia kapasitas Anda

Praktik terbaik untuk Instans Terkelola Lambda

Konfigurasi penyedia kapasitas

Pisahkan penyedia kapasitas berdasarkan tingkat kepercayaan. Buat penyedia kapasitas yang berbeda untuk beban kerja dengan persyaratan keamanan yang berbeda. Semua Fungsi Acara Lambda yang ditetapkan untuk penyedia kapasitas yang sama harus saling dipercaya, karena penyedia kapasitas berfungsi sebagai batas keamanan.

Gunakan nama deskriptif. Sebutkan penyedia kapasitas untuk secara jelas menunjukkan tujuan penggunaan dan tingkat kepercayaan mereka (misalnya,`production-trusted`,`dev-sandbox`). Ini membantu tim memahami tujuan dan postur keamanan masing-masing penyedia kapasitas.

Gunakan beberapa Availability Zone. Tentukan subnet di beberapa Availability Zone saat membuat penyedia kapasitas. Lambda meluncurkan tiga instans secara default untuk ketahanan AZ, memastikan ketersediaan tinggi untuk fungsi Anda.

Pemilihan tipe instans

Biarkan Lambda memilih jenis instance. Secara default, Lambda memilih jenis instans terbaik untuk beban kerja Anda. Sebaiknya izinkan Instans Terkelola Lambda memilih jenis instans untuk Anda, karena membatasi jumlah kemungkinan jenis instans dapat mengakibatkan ketersediaan lebih rendah.

Tentukan jenis instance untuk persyaratan tertentu. Jika Anda memiliki persyaratan perangkat keras tertentu, setel jenis instans yang diizinkan ke daftar instance yang kompatibel. Contoh:

- Untuk aplikasi yang membutuhkan bandwidth jaringan tinggi, pilih beberapa jenis instans n
- Untuk lingkungan pengujian atau pengembangan dengan batasan biaya, pilih jenis instance yang lebih kecil seperti m7a.large

Konfigurasi fungsi

Pilih pengaturan memori dan vCPU yang sesuai. Pilih konfigurasi memori dan vCPU yang mendukung eksekusi multi-konkuren fungsi Anda. Ukuran fungsi minimum yang didukung adalah 2GB dan 1 vCPU.

- Untuk aplikasi Python, pilih rasio memori yang lebih tinggi terhadap vCPU (seperti 4 banding 1 atau 8 banding 1) karena cara Python menangani multi-konkurensi

- Untuk CPU-intensive operasi atau fungsi yang melakukan IO kecil, pilih lebih dari satu vCPU
- Untuk IO-heavy aplikasi seperti layanan web atau pekerjaan batch, multi-konkurensi memberikan manfaat paling besar

Konfigurasikan konkurensi maksimum dengan tepat. Lambda memilih default yang masuk akal untuk konkurensi maksimum yang menyeimbangkan konsumsi sumber daya dan throughput. Sesuaikan pengaturan ini berdasarkan penggunaan sumber daya fungsi Anda:

- Tingkatkan konkurensi maksimum (hingga 64 per vCPU) jika pemanggilan fungsi Anda menggunakan CPU yang sangat sedikit
- Kurangi konkurensi maksimum jika aplikasi Anda mengkonsumsi sejumlah besar memori dan sangat sedikit CPU

Perhatikan bahwa lingkungan eksekusi dengan konkurensi yang sangat rendah mungkin mengalami throttle dan kesulitan penskalaan.

Long-running fungsi

Fungsi pada Instans AWS Lambda Terkelola dapat berjalan hingga 90 menit (5.400 detik) per pemanggilan untuk pemanggilan asinkron dan untuk pemanggilan pemetaan sumber peristiwa, kecuali pemetaan sumber peristiwa Amazon MQ dan Amazon DocumentDB, yang tetap dibatasi hingga 15 menit. Pemanggilan sinkron dan fase inisialisasi fungsi juga tetap terbatas hingga 15 menit. Untuk informasi selengkapnya tentang menyetel batas waktu, lihat Mengonfigurasi batas waktu fungsi Lambda. Karena fungsi Anda dapat berjalan lebih lama, tinjau pertimbangan berikut untuk komponen yang bersifat fana, seperti koneksi jaringan dan kredensil.

Mengakomodasi batas waktu koneksi idle. Pastikan batas waktu koneksi idle pada layanan hilir seperti Amazon RDS, ElastiCache Amazon, dan API eksternal mengakomodasi durasi fungsi penuh. Jika fungsi Anda merutekan lalu lintas melalui gateway NAT, kirim paket keep-alive untuk mencegah koneksi idle terputus setelah batas waktu idle gateway NAT 350 detik.

Hormati nilai DNS TTL. Hormati nilai TTL DNS saat Anda menyelesaikan nama host eksternal. AWS SDK menangani ini secara otomatis, tetapi klien HTTP khusus mungkin menyimpan resolusi DNS di luar TTL mereka.

Segarkan kredensi sementara. Jika fungsi Anda memperoleh kredensi atau token sementara, verifikasi bahwa token tersebut tetap valid selama durasi eksekusi penuh, atau segarkan kembali di latar belakang.

Desain untuk idempotensi. Lambda tidak menjamin pemrosesan yang tepat sekali. Saat fungsi berjalan lebih lama, jendela untuk percobaan ulang dan pengiriman duplikat meningkat. Gunakan Powertools untuk AWS Lambda untuk mengimplementasikan idempotensi untuk operasi seperti pembayaran atau penulisan database, sehingga mereka menghasilkan hasil yang sama bahkan jika mereka berjalan lebih dari sekali. Jika Anda menggunakan fungsi tahan lama Lambda, langkah-langkah memiliki semantik eksekusi paling sedikit sekali: SDK eksekusi yang tahan lama melewatkan langkah-langkah yang telah selesai selama pemutaran ulang, tetapi langkah-langkah yang gagal sebelum checkpointing dapat berjalan lagi. Anda dapat menggunakan nama eksekusi sebagai kunci idempotensi.

Tune pemetaan sumber acara untuk pemrosesan yang lebih lama. Untuk Amazon SQS, setel batas waktu visibilitas antrian setidaknya enam kali batas waktu fungsi, sehingga Lambda memiliki cukup waktu untuk mencoba kembali batch jika fungsi dibatasi. Lambda memvalidasi ini saat Anda membuat pemetaan sumber peristiwa, tetapi tidak mencegah perubahan selanjutnya pada antrian atau fungsi yang membuat ketidakcocokan. Untuk Amazon Kinesis Data Streams dan Amazon DynamoDB Streams, konfigurasikan jendela batching maksimum dan faktor paralelisasi untuk memperhitungkan waktu pemrosesan yang lebih lama per batch, dan aktifkan pelaporan kegagalan batch sebagian sehingga hanya catatan gagal yang dicoba ulang, bukan seluruh batch.

Konfigurasi penskalaan

Tetapkan pemanfaatan sumber daya target yang tepat. Secara default, Lambda mempertahankan ruang kepala yang cukup untuk lalu lintas Anda berlipat ganda dalam waktu 5 menit tanpa throttle. Sesuaikan ini berdasarkan karakteristik beban kerja Anda:

- Untuk beban kerja yang sangat stabil atau aplikasi yang tidak sensitif terhadap throttle, tetapkan target ke tingkat tinggi untuk mencapai pemanfaatan yang lebih tinggi dan biaya yang lebih rendah
- Untuk beban kerja dengan potensi ledakan lalu lintas, tetapkan target sumber daya ke tingkat rendah untuk mempertahankan ruang kepala tambahan

Rencanakan pertumbuhan lalu lintas. Jika lalu lintas Anda lebih dari dua kali lipat dalam 5 menit, Anda mungkin melihat throttle saat Lambda meningkatkan instance dan lingkungan eksekusi. Rancang aplikasi Anda untuk menangani potensi pelambatan selama periode peningkatan skala yang cepat.

Keamanan

Terapkan hak istimewa paling sedikit untuk PassCapacityProvider izin. Berikan `lambda:PassCapacityProvider` izin hanya untuk penyedia kapasitas yang diperlukan. Gunakan izin tingkat sumber daya untuk membatasi penyedia kapasitas mana yang dapat ditetapkan pengguna ke fungsi.

Pantau penggunaan penyedia kapasitas. Gunakan AWS CloudTrail untuk memantau tugas penyedia kapasitas dan pola akses. Ini membantu mengidentifikasi upaya akses yang tidak sah dan memastikan kepatuhan terhadap kebijakan keamanan.

Pisahkan beban kerja yang tidak tepercaya. Jangan mengandalkan kontainer untuk isolasi keamanan antara beban kerja yang tidak tepercaya. Gunakan penyedia kapasitas yang berbeda untuk memisahkan beban kerja yang tidak saling dipercaya.

Optimalisasi biaya

Gunakan opsi harga EC2. Manfaatkan EC2 Savings Plans dan Instans Cadangan untuk mengurangi biaya. Opsi harga ini berlaku untuk komputasi EC2 yang mendasarinya (biaya manajemen 15% tidak didiskon).

Optimalkan untuk beban kerja kondisi mapan. Instans Terkelola Lambda paling cocok untuk fungsi kondisi mapan dengan lalu lintas volume tinggi yang dapat diprediksi. Untuk pola lalu lintas yang meledak, Lambda (default) mungkin lebih hemat biaya.

Pantau pemanfaatan sumber daya. Lacak CloudWatch metrik untuk memahami pemanfaatan CPU dan memori. Sesuaikan alokasi memori fungsi dan pemilihan jenis instans berdasarkan pola penggunaan aktual untuk mengoptimalkan biaya.

Pemantauan dan observabilitas

Pantau metrik penyedia kapasitas. Melacak metrik tingkat penyedia kapasitas termasuk CPUUutilization,, VCPUAvailable MemoryUtilization, dan MemoryAvailable untuk memverifikasi sumber daya yang memadai tersedia untuk beban kerja Anda.

Pantau metrik lingkungan eksekusi. Lacak metrik tingkat lingkungan eksekusi termasuk ExecutionEnvironmentConcurrency dan ExecutionEnvironmentConcurrencyLimit untuk memahami perilaku penskalaan dan mengidentifikasi potensi pelambatan.

Mengatur CloudWatch alarm. Buat CloudWatch alarm untuk metrik utama untuk mengidentifikasi masalah secara proaktif:

- Pemanfaatan CPU atau memori yang tinggi
- Kapasitas yang tersedia rendah
- Mendekati batas konkurensi

Language-specific pertimbangan

Ikuti praktik terbaik khusus bahasa. Setiap bahasa pemrograman menangani multi-konkurensi secara berbeda. Tinjau panduan khusus bahasa untuk rekomendasi terperinci:

- Java: Gunakan koleksi aman utas,`AtomicInteger`, dan `ThreadLocal` untuk status khusus permintaan
- Node.js: Gunakan InvokeStore untuk semua status permintaan khusus dan hindari variabel global
- Python: Gunakan nama file unik `/tmp` dengan ID permintaan dan pertimbangkan isolasi memori berbasis proses
- Rust: Gunakan `run_concurrent` sebagai gantinya`run`, dengan `concurrency-tokio` fitur diaktifkan. Pawang harus `Clone` `+Send`.

Uji keamanan benang dan masalah konkurensi. Sebelum menerapkan ke produksi, uji fungsi Anda secara menyeluruh untuk masalah keselamatan benang, kondisi balapan, dan isolasi status yang tepat di bawah beban bersamaan.

Langkah selanjutnya

- Pelajari tentang penyedia kapasitas untuk Instans Terkelola Lambda
- Memahami penskalaan untuk Instans Terkelola Lambda
- Tinjau panduan khusus runtime untuk Java, Node.js, dan Python
- Konfigurasikan konektivitas VPC untuk penyedia kapasitas Anda
- Pantau Instans Terkelola Lambda dengan metrik CloudWatch

Memecahkan Masalah Instans Terkelola Lambda

Masalah pelambatan dan penskalaan

Pelambatan sesekali

Masalah: Anda mengalami kesalahan pelambatan (HTTP 429) selama operasi normal.

Penyebab: Inst ans Terkelola Lambda mungkin menolak panggilan baru untuk melindungi panggilan yang sudah dalam penerbangan. Jika lingkungan eksekusi Anda memiliki pemanfaatan tinggi secara konsisten, pemanggilan baru mungkin dibatasi.

Solusi:

- Memantau metrik penskalaan: Per iksa grafik Throttle Reasons untuk memahami alasan pembatasan dan masalah penskalaan kapasitas. Dalam contoh berikut, CPU tinggi menyebabkan throttle.
- Tinjau konfigurasi fungsi: Pastikan memori fungsi dan pengaturan vCPU mendukung eksekusi multi-bersamaan. Tingkatkan memori fungsi atau alokasi vCPU jika diperlukan. Jika `ExecutionEnvironmentVCPUUtilization` tinggi, coba tambahkan lebih banyak vCPU per fungsi.

  ```
  aws lambda update-function \
    --function-name my-function \
    ...
    --memory-size 8192
  ```

  Jika `ExecutionEnvironmentMemoryUtilization` tinggi, coba tambahkan lebih banyak memori per vCPU:

  ```
  aws lambda update-function \
    --function-name my-function \
    --capacity-provider-config '{
      "LambdaManagedInstancesCapacityProviderConfig": {
        "ExecutionEnvironmentMemoryGiBPerVCpu": 4.0
      }
    }'
  ```

Throttle selama peningkatan skala

Masalah: Anda mengalami kesalahan pelambatan (HTTP 429) ketika lalu lintas meningkat dengan cepat.

Penyebab: Lambda Managed Instances menskalakan secara asinkron berdasarkan pemanfaatan sumber daya CPU dan saturasi multi-konkurensi. Jika lalu lintas Anda lebih dari dua kali lipat dalam waktu 5 menit, Anda mungkin melihat pembatasan saat Lambda meningkatkan instans dan lingkungan eksekusi untuk memenuhi permintaan.

Solusi:

- Sesuaikan pemanfaatan sumber daya target: Jika beban kerja Anda memiliki pola lalu lintas yang dapat diprediksi, tetapkan pemanfaatan sumber daya target yang lebih rendah untuk mempertahankan ruang tambahan untuk ledakan lalu lintas.

```
aws lambda create-capacity-provider \
  --name my-capacity-provider \
  ...
  --capacity-provider-scaling-config '{
    "ScalingMode": "Manual",
    "ScalingPolicies": [
      {
        "PredefinedMetricType": "LambdaCapacityProviderAverageCPUUtilization",
        "TargetValue": 30.0
      }
    ]
  }'
```

- Pre-warm kapasitas: Untuk peningkatan lalu lintas yang direncanakan, gunakan `PutFunctionScalingConfig` API untuk melakukan pra-pemanasan kapasitas tambahan.

```
aws lambda put-function-scaling-config \
  --function-name my-function \
  --qualifier 1 \
  --function-scaling-config '{
    "MinExecutionEnvironments": 100
  }'
```

Menurunkan skala lambat

Masalah: Instans membutuhkan waktu lama untuk diturunkan setelah lalu lintas berkurang.

Penyebab: Lambda Managed Instances akan menurunkan skala secara bertahap untuk menjaga ketersediaan dan menghindari perubahan kapasitas yang cepat yang dapat memengaruhi kinerja.

Solusi:

Ini adalah perilaku yang diharapkan. Lambda mengurangi instance secara konservatif untuk memastikan stabilitas. Pantau CloudWatch metrik Anda untuk melacak jumlah instans yang sedang berjalan.

Masalah konkurensi

Lingkungan eksekusi dengan pengalaman konkurensi rendah membatasi

Masalah: Fungsi Anda mengalami pelambatan meskipun memiliki kapasitas yang tersedia.

Penyebab: Lingkungan eksekusi dengan konkurensi maksimum yang sangat rendah mungkin mengalami kesulitan melakukan penskalaan secara efektif. Instans Terkelola Lambda dirancang untuk aplikasi multi-serentak.

Solusi:

- Tingkatkan konkurensi maksimum: Jika pemanggilan fungsi Anda menggunakan CPU yang sangat sedikit, tingkatkan pengaturan konkurensi maksimum hingga 64 per vCPU.

```
aws lambda update-function \
  --function-name ordering-api-backend \
  --capacity-provider-config '{
    "LambdaManagedInstancesCapacityProviderConfig": {
      "PerExecutionEnvironmentMaxConcurrency": 32
    }
  }'
```

- Optimalkan kode fungsi: T injau kode fungsi Anda untuk mengurangi konsumsi CPU per pemanggilan, memungkinkan konkurensi yang lebih tinggi.
- Sesuaikan memori fungsi dan vCPU: Pastikan fungsi Anda memiliki sumber daya yang cukup untuk menangani beberapa pemanggilan bersamaan.

Masalah keamanan utas (runtime Java)

Masalah: Fungsi Java Anda menghasilkan hasil yang salah atau mengalami kondisi balapan di bawah beban.

Penyebab: Beberapa utas mengeksekusi metode handler secara bersamaan, dan status bersama tidak aman untuk utas.

Solusi:

- Gunakan `AtomicInteger` atau `AtomicLong` untuk penghitung alih-alih tipe primitif
- Ganti `HashMap` dengan `ConcurrentHashMap`

- Gunakan `Collections.synchronizedList()` untuk membungkus `ArrayList`
- Gunakan `ThreadLocal` untuk status khusus permintaan
- Akses ID jejak dari objek Lambda Context, bukan variabel lingkungan

Untuk panduan terperinci, lihat dokumentasi runtime Java untuk Lambda Managed Instances.

Masalah isolasi status (Node.js runtime)

Masalah: Node.js Fungsi Anda mengembalikan data dari permintaan yang berbeda atau mengalami kerusakan data.

Penyebab: Vari abel global dibagikan di seluruh pemanggilan bersamaan pada utas pekerja yang sama. Ketika operasi asinkron menghasilkan kontrol, pemanggilan lain dapat memodifikasi status bersama.

Solusi:

- Instal dan gunakan `@aws/lambda-invoke-store` untuk semua status khusus permintaan
- Ganti variabel global dengan `InvokeStore.set()` dan `InvokeStore.get()`
- Gunakan nama file unik `/tmp` dengan ID permintaan
- Akses ID jejak menggunakan al `InvokeStore.getXRayTraceId()` ih-alih variabel lingkungan

Untuk panduan terperinci, lihat dokumentasi Node.js runtime untuk Lambda Managed Instances.

Konflik file (runtime Python)

Masalah: Fungsi Python Anda membaca data yang salah dari file di`/tmp`.

Penyebab: Beberapa proses berbagi `/tmp` direktori. Menulis secara bersamaan ke file yang sama dapat menyebabkan kerusakan data.

Solusi:

- Gunakan nama file unik dengan ID permintaan: `/tmp/request_{context.request_id}.txt`
- Gunakan penguncian file dengan `fcntl.flock()` untuk file bersama
- Bersihkan file sementara `os.remove()` setelah digunakan

Untuk panduan terperinci, lihat dokumentasi runtime Python untuk Lambda Managed Instances.

Masalah kinerja

Pemanfaatan memori tinggi

Masalah: Fungsi Anda mengalami pemanfaatan memori yang tinggi atau kesalahan kehabisan memori.

Penyebab: Setiap permintaan bersamaan di Python berjalan dalam proses terpisah dengan ruang memori sendiri. Total penggunaan memori sama dengan memori per proses dikalikan dengan proses bersamaan.

Solusi:

- Pantau `MemoryUtilization` metrik di CloudWatch
- Kurangi `MaxConcurrency` pengaturan jika penggunaan memori mendekati batas memori fungsi
- Tingkatkan alokasi memori fungsi untuk mendukung konkurensi yang lebih tinggi
- Optimalkan penggunaan memori dengan memuat data sesuai permintaan alih-alih selama inisialisasi

Kinerja yang tidak konsisten

Masalah: Kin erja fungsi bervariasi secara signifikan antara pemanggilan.

Penyebab: Lambda mungkin memilih jenis instans yang berbeda berdasarkan ketersediaan, atau fungsi mungkin berjalan pada instans dengan ketersediaan sumber daya yang bervariasi.

Solusi:

- Tentukan jenis instans yang diizinkan: Jika Anda memiliki persyaratan kinerja tertentu, konfigurasikan jenis instans yang diizinkan di penyedia kapasitas Anda untuk membatasi jenis instans yang dapat dipilih Lambda.
- Memantau metrik tingkat instance: Lacak `CPUUtilization` dan `MemoryUtilization` pada tingkat penyedia kapasitas untuk mengidentifikasi kendala sumber daya.
- Tinjau metrik kapasitas: **`vCPUAvailable`** Per iksa dan `MemoryAvailable` pastikan sumber daya yang cukup tersedia di instans Anda.

Masalah penyedia kapasitas

Versi fungsi gagal menjadi AKTIF

Masalah: Versi fungsi Anda tetap dalam keadaan tertunda setelah penerbitan.

Penyebab: Lambda meluncurkan Instans Terkelola dan memulai lingkungan eksekusi. Proses ini membutuhkan waktu, terutama untuk versi fungsi pertama pada penyedia kapasitas baru.

Solusi:

Tunggu Lambda menyelesaikan proses inisialisasi. Lambda meluncurkan tiga instance secara default untuk ketahanan AZ dan memulai tiga lingkungan eksekusi sebelum menandai versi fungsi Anda AKTIF. Ini biasanya memerlukan waktu beberapa menit.

Tidak dapat menghapus penyedia kapasitas

Masalah: Anda menerima kesalahan saat mencoba menghapus penyedia kapasitas.

Penyebab: Anda tidak dapat menghapus penyedia kapasitas yang memiliki versi fungsi yang dilampirkan padanya.

Solusi:

1. Identifikasi semua versi fungsi menggunakan penyedia kapasitas dengan `ListFunctionVersionsByCapacityProvider` API.
2. Hapus atau perbarui versi fungsi tersebut untuk menghapus asosiasi penyedia kapasitas.
3. Coba lagi menghapus penyedia kapasitas.

Pesan kesalahan umum selama penerbitan fungsi

Masalah: Anda menemukan pesan kesalahan umum seperti “Terjadi kesalahan internal selama penerbitan” saat menerbitkan fungsi.

Solusi:

- Periksa izin IAM: Pastikan Anda memiliki `lambda:PassCapacityProvider` izin untuk penyedia kapasitas yang Anda coba gunakan.
- Verifikasi konfigurasi penyedia kapasitas: Konfirmasikan bahwa penyedia kapasitas Anda berada dalam status AKTIF menggunakan `GetCapacityProvider` API.

- Tinjau konfigurasi VPC: Pastikan subnet dan grup keamanan yang ditentukan di penyedia kapasitas Anda dikonfigurasi dan dapat diakses dengan benar.
- Periksa AWS CloudTrail log: Tinjau CloudTrail log untuk informasi kesalahan terperinci tentang operasi yang gagal.

Masalah pemantauan dan pengamatan

CloudWatch Metrik yang hilang

Masalah: Anda tidak melihat metrik yang diharapkan CloudWatch untuk penyedia kapasitas atau fungsi Anda.

Penyebab: Metrik dipublikasikan pada interval 1 menit. Penyedia kapasitas atau fungsi baru mungkin tidak memiliki metrik yang tersedia segera.

Solusi:

Tunggu setidaknya 5-10 menit setelah menerbitkan versi fungsi sebelum mengharapkan metrik muncul di CloudWatch. Pastikan Anda melihat namespace (`AWS/Lambda`) dan dimensi (`CapacityProviderName`,`FunctionName`, atau`InstanceType`) yang benar.

Tidak dapat menemukan CloudWatch log

Masalah: Fungsi Anda berhasil dijalankan, tetapi Anda tidak dapat menemukan CloudWatch log di Log.

Penyebab: Inst ans Terkelola Lambda berjalan di VPC Anda dan memerlukan konektivitas jaringan untuk mengirim log ke CloudWatch Log. Tanpa konfigurasi konektivitas VPC yang tepat, fungsi Anda tidak dapat mencapai titik akhir layanan CloudWatch Logs.

Solusi:

Konfigurasikan konektivitas VPC untuk mengaktifkan fungsi Anda mengirim log ke CloudWatch Log. Anda memiliki tiga opsi:

Opsi 1: Titik akhir VPC untuk CloudWatch Log (direkomendasikan untuk produksi)

1. Buka konsol Amazon VPC di console.aws.amazon. com/vpc/.
2. Di panel navigasi, pilih Titik Akhir.
3. Pilih Buat Titik Akhir.
4. Untuk kategori Layanan, pilih AWS layanan.

5. Untuk Nama Layanan, pilih `com.amazonaws.region.logs` (ganti `region` dengan Wil AWS ayah Anda).
6. Untuk VPC, pilih VPC yang digunakan oleh penyedia kapasitas Anda.
7. Untuk Subnet, pilih subnet tempat Anda ingin membuat antarmuka jaringan titik akhir. Untuk mendapatkan ketersediaan tinggi, pilih subnet di beberapa Zona Ketersediaan.
8. Untuk Grup keamanan, pilih grup keamanan yang mengizinkan lalu lintas HTTPS masuk (port 443) dari grup keamanan fungsi Anda.
9. Aktif kan DNS Pri badi untuk titik akhir.
10. Pilih Buat titik akhir.

Opsi 2: Subnet publik dengan gateway internet

Jika penyedia kapasitas Anda menggunakan subnet publik, pastikan:

1. Gateway internet terpasang ke VPC Anda
2. Tabel rute merutekan `0.0.0.0/0` lalu lintas ke gateway internet
3. Grup keamanan mengizinkan lalu lintas HTTPS keluar pada port 443

Opsi 3: Subnet pribadi dengan gateway NAT

Jika penyedia kapasitas Anda menggunakan subnet pribadi, pastikan:

1. Gerbang NAT ada di subnet publik
2. Tabel rute subnet pribadi merutekan `0.0.0.0/0` lalu lintas ke gateway NAT
3. Tabel rute subnet publik merutekan `0.0.0.0/0` lalu lintas ke gateway internet
4. Grup keamanan mengizinkan lalu lintas HTTPS keluar pada port 443

Untuk panduan terperinci tentang opsi konektivitas VPC, lihat Konektivitas VPC untuk Instans Terkelola Lambda.

Kesulitan menghubungkan log dari permintaan bersamaan

Masalah: Log dari permintaan yang berbeda disisipkan, sehingga sulit untuk melacak permintaan individu.

Penyebab: Inter leaving log diharapkan dan perilaku standar dalam sistem multi-bersamaan.

Solusi:

- Gunakan logging terstruktur dengan format JSON: Sertakan ID permintaan di semua pernyataan log
- Java: Gunakan Log4j dengan `ThreadContext` untuk secara otomatis menyertakan ID permintaan
- Node.js: Gunakan `console.log()` dengan pemformatan JSON dan sertakan `InvokeStore.getRequestId()`
- Python: Gunakan modul logging standar dengan pemformatan JSON dan sertakan `context.request_id`

Untuk panduan terperinci, lihat halaman dokumentasi khusus runtime.

Mendapatkan bantuan tambahan

Jika Anda terus mengalami masalah setelah mencoba solusi ini:

1. Tinjau CloudWatch metrik: Periksa penyedia kapasitas dan metrik lingkungan eksekusi untuk mengidentifikasi kendala sumber daya atau masalah penskalaan.
2. Periksa AWS CloudTrail log: Tinjau CloudTrail log untuk informasi terperinci tentang panggilan dan kesalahan API.
3. Hu AWS bungi Dukungan: Jika Anda tidak dapat menyelesaikan masalah, AWS hubungi Dukungan dengan rincian tentang konfigurasi penyedia kapasitas, konfigurasi fungsi, dan pesan kesalahan spesifik yang Anda temui.

Langkah selanjutnya

- Pelajari tentang penyedia kapasitas untuk Instans Terkelola Lambda
- Memahami penskalaan untuk Instans Terkelola Lambda
- Tinjau panduan khusus runtime untuk Java, Node.js, dan Python ???
- Memantau Instans Terkelola Lambda dengan metrik CloudWatch
- Tinjau praktik terbaik untuk Instans Terkelola Lambda

Mengelola dependensi Fungsi Acara Lambda dengan lapisan

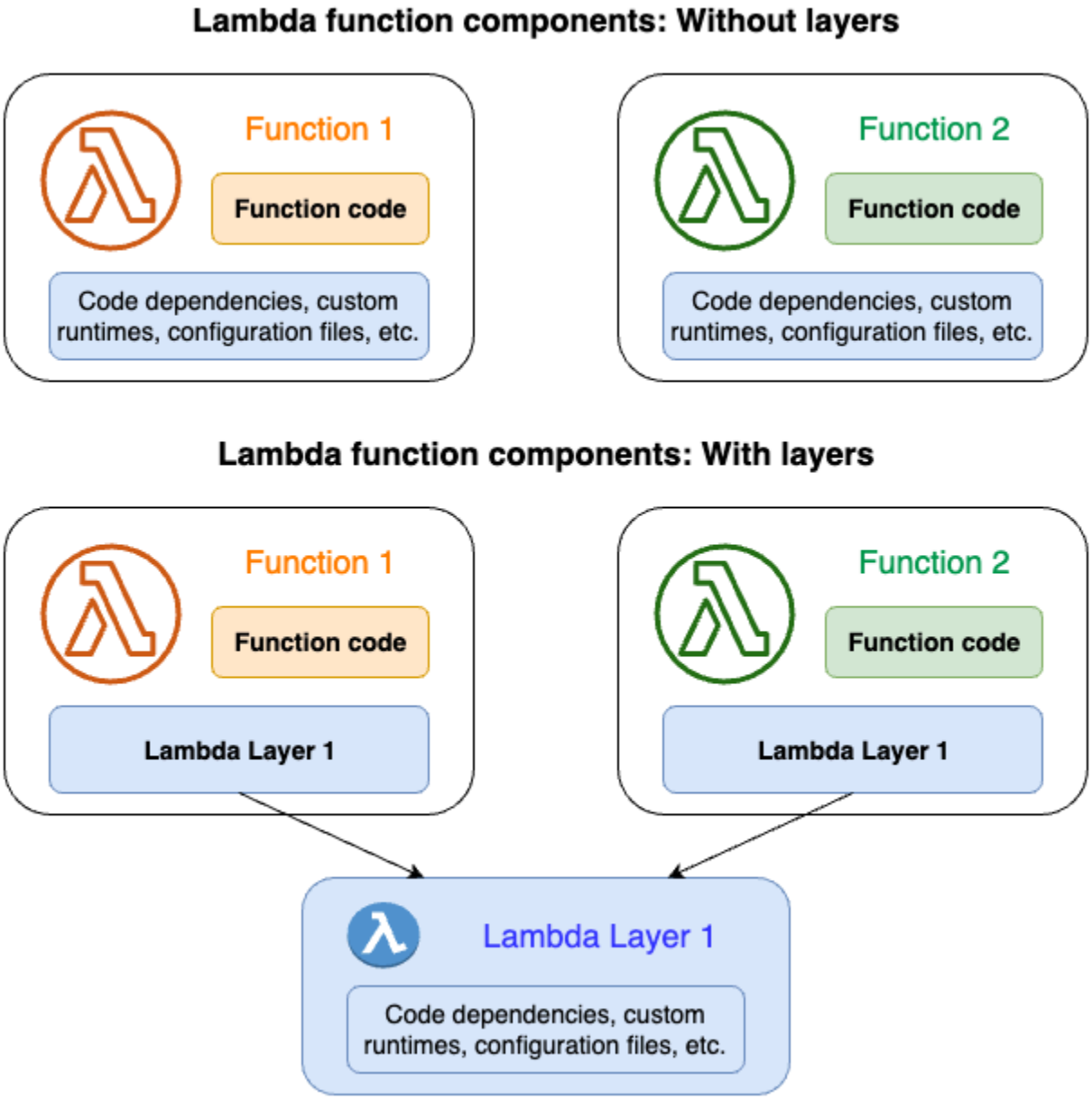
Lapisan Lambda adalah arsip file.zip yang berisi kode atau data tambahan. Lapisan biasanya berisi dependensi pustaka, runtime kustom, atau file konfigurasi.

Ada beberapa alasan mengapa Anda mungkin mempertimbangkan untuk menggunakan lapisan:

- Untuk mengurangi ukuran paket penyebaran Anda. Alih-alih menyertakan semua dependensi Event Function Anda bersama dengan kode Event Function Anda dalam paket deployment Anda, letakkan di layer. Ini membuat paket penerapan tetap kecil dan terorganisir.
- Untuk memisahkan logika fungsi inti dari dependensi. Dengan layer, Anda dapat memperbarui dependensi Event Function Anda independen dari kode Event Function Anda, dan sebaliknya. Ini mempromosikan pemisahan kekhawatiran dan membantu Anda fokus pada logika Fungsi Acara Anda.
- Untuk berbagi dependensi di beberapa fungsi. Setelah Anda membuat layer, Anda dapat menerapkannya ke sejumlah Fungsi Acara di akun Anda. Tanpa lapisan, Anda perlu menyertakan dependensi yang sama di setiap paket penerapan individu.
- Untuk menggunakan editor kode konsol Lambda. Editor kode adalah alat yang berguna untuk menguji pembaruan kode Fungsi Acara minor dengan cepat. Namun, Anda tidak dapat menggunakan editor jika ukuran paket penerapan Anda terlalu besar. Menggunakan lapisan mengurangi ukuran paket Anda dan dapat membuka kunci penggunaan editor kode.
- Untuk mengunci versi SDK yang disematkan. SDK yang disematkan mungkin berubah tanpa pemberitahuan saat AWS merilis layanan dan fitur baru. Anda dapat mengunci versi SDK dengan membuat layer Lambda dengan versi spesifik yang diperlukan. Fungsi Acara kemudian selalu menggunakan versi di lapisan, bahkan jika versi yang disematkan dalam layanan berubah.

Jika Anda bekerja dengan Lambda Event Functions di Go atau Rust, sebaiknya jangan menggunakan layer. Untuk Go dan Rust Event Functions, Anda memberikan kode Event Function Anda sebagai executable, yang menyertakan kode Event Function yang dikompilasi bersama dengan semua dependensinya. Menempatkan dependensi Anda di lapisan memaksa Fungsi Peristiwa Anda memuat rakitan tambahan secara manual selama fase inisialisasi, yang dapat meningkatkan waktu mulai dingin. Untuk kinerja optimal untuk Go dan Rust Event Functions, sertakan dependensi Anda bersama dengan paket penerapan Anda.

Diagram berikut menggambarkan perbedaan arsitektur tingkat tinggi antara dua Fungsi Peristiwa yang berbagi dependensi. Satu menggunakan lapisan Lambda, dan yang lainnya tidak.



Saat Anda menambahkan layer ke Fungsi Peristiwa, Lambda mengekstrak konten layer ke dalam `/opt` direktori di lingkungan eksekusi Event Function Anda. Semua runtime Lambda yang didukung secara native menyertakan jalur ke direktori tertentu di dalam direktori. `/opt` Ini memberi Event Function Anda akses ke konten layer Anda. Untuk informasi selengkapnya tentang jalur spesifik ini dan cara mengemas layer Anda dengan benar, lihatthe section called “Lapisan kemasan”.

Anda dapat menyertakan hingga lima lapisan per Fungsi Acara. Selain itu, Anda dapat menggunakan layer hanya dengan Lambda Event Functions yang digunakan sebagai arsip file.zip. Untuk Fungsi Acara yang didefinisikan sebagai gambar kontainer, paketkan runtime pilihan Anda dan semua

dependensi kode saat Anda membuat gambar kontainer. Untuk informasi selengkapnya, lihat Bekerja dengan lapisan dan ekstensi Lambda dalam gambar kontainer di Blog AWS Komputasi.

Topik

- Cara menggunakan lapisan
- Versi lapisan dan lapisan
- Kemasan konten lapisan Anda
- Membuat dan menghapus layer di Lambda
- Menambahkan lapisan ke Fungsi Acara
- Penggunaan AWS CloudFormation dengan lapisan
- Penggunaan AWS SAM dengan lapisan

Cara menggunakan lapisan

Untuk membuat layer, paketkan dependensi Anda ke dalam file.zip, mirip dengan cara Anda membuat paket penerapan normal. Lebih khusus lagi, proses umum membuat dan menggunakan lapisan melibatkan tiga langkah ini:

- Pertama, paketkan konten layer Anda. Ini berarti membuat arsip file.zip. Untuk informasi selengkapnya, lihat the section called “Lapisan kemasan”.
- Selanjutnya, buat layer di Lambda. Untuk informasi selengkapnya, lihat the section called “Membuat dan menghapus lapisan”.
- Tambahkan layer ke Event Function (s) Anda. Untuk informasi selengkapnya, lihat the section called “Menambahkan lapisan”.

Versi lapisan dan lapisan

Versi lapisan adalah snapshot yang tidak dapat diubah dari versi tertentu dari lapisan. Saat Anda membuat layer baru, Lambda membuat versi layer baru dengan nomor versi 1. Setiap kali Anda mempublikasikan pembaruan ke layer, Lambda menambah nomor versi dan membuat versi layer baru.

Setiap versi layer diidentifikasi oleh Amazon Resource Name (ARN) yang unik. Saat menambahkan layer ke Fungsi Acara, Anda harus menentukan versi layer yang tepat yang ingin Anda gunakan (misalnya,`arn:aws:lambda:us-east-1:123456789012:layer:my-layer:1`).

Kemasan konten lapisan Anda

Lapisan Lambda adalah arsip file.zip yang berisi kode atau data tambahan. Lapisan biasanya berisi dependensi pustaka, runtime kustom, atau file konfigurasi.

Bagian ini menjelaskan cara mengemas konten layer Anda dengan benar. Untuk informasi konseptual lebih lanjut tentang lapisan dan mengapa Anda mungkin mempertimbangkan untuk menggunakannya, lihatthe section called “Lapisan”.

Langkah pertama untuk membuat layer adalah untuk menggabungkan semua konten layer Anda ke dalam arsip file.zip. Karena fungsi Lambda berjalan di Amazon Linux, konten layer Anda harus dapat dikompilasi dan dibangun di lingkungan Linux.

Untuk memastikan bahwa konten lapisan Anda berfungsi dengan baik di lingkungan Linux, kami sarankan untuk membuat konten lapisan Anda menggunakan alat seperti Docker.

Topik

- Jalur lapisan untuk setiap runtime Lambda

Jalur lapisan untuk setiap runtime Lambda

Saat Anda menambahkan lapisan ke fungsi, Lambda memuat konten lapisan ke dalam `/opt` direktori lingkungan eksekusi itu. Untuk setiap runtime Lambda, `PATH` variabel sudah menyertakan jalur folder tertentu dalam direktori. `/opt` Untuk memastikan bahwa Lambda mengambil konten layer Anda, file layer.zip Anda harus memiliki dependensinya di salah satu jalur folder berikut:

Runtime	Jalan
Node.js	`nodejs/node_modules`
	`nodejs/node18/node_modules` (`NODE_PATH`)
	`nodejs/node20/node_modules` (`NODE_PATH`)
	`nodejs/node22/node_modules` (`NODE_PATH`)
Python	`python`

Runtime	Jalan
	`python/lib/ python3.x /site-pac kages` (direktori situs)
Java	`java/lib` (`CLASSPATH`)
Ruby	`ruby/gems/3.4.0` (`GEM_PATH`)
	`ruby/lib` (`RUBYLIB`)
Runtime	`bin` (`PATH`)
	`lib` (`LD_LIBRARY_PATH`)

Contoh berikut menunjukkan cara Anda dapat menyusun folder dalam arsip .zip lapisan Anda.

Node.js

Example struktur file untuk AWS X-Ray SDK untuk Node.js

```
xray-sdk.zip
# nodejs/node_modules/aws-xray-sdk
```

Python

Example

```
python/              # Required top-level directory
### requests/
### boto3/
### numpy/
### (dependencies of the other packages)
```

Ruby

Example struktur file untuk JSON gem

```
json.zip
# ruby/gems/3.4.0/
```

```
                | build_info
                | cache
                | doc
                | extensions
                | gems
                | # json-2.1.0
                # specifications
                  # json-2.1.0.gemspec
```

Java

Example struktur file untuk file JAR Jackson

```
layer_content.zip
# java
    # lib
        # jackson-core-2.17.0.jar
        # <other potential dependencies>
        # ...
```

All

Example struktur file untuk pustaka jq

```
jq.zip
# bin/jq
```

Untuk instruksi khusus bahasa tentang pengemasan, pembuatan, dan penambahan lapisan, lihat halaman berikut:

- Node.js – the section called “Lapisan”
- Python – the section called “Lapisan”
- Ruby – the section called “Lapisan”
- Jawa — the section called “Lapisan”

Kami merekomendasikan agar tidak menggunakan lapisan untuk mengelola dependensi untuk Lambda Event Functions yang ditulis dalam Go dan Rust. Ini karena Lambda Event Functions yang ditulis dalam bahasa-bahasa ini dikompilasi menjadi satu executable, yang Anda berikan kepada

Lambda ketika Anda menerapkan Fungsi Acara Anda. Executable ini berisi kode Event Function Anda yang dikompilasi, bersama dengan semua dependensinya. Menggunakan lapisan tidak hanya mempersulit proses ini, tetapi juga menyebabkan peningkatan waktu mulai dingin karena Fungsi Acara Anda perlu memuat rakitan ekstra secara manual ke dalam memori selama fase init.

Untuk menggunakan dependensi eksternal dengan Go dan Rust Lambda Event Functions, sertakan mereka langsung dalam paket penerapan Anda.

Membuat dan menghapus layer di Lambda

Lapisan Lambda adalah arsip file.zip yang berisi kode atau data tambahan. Lapisan biasanya berisi dependensi pustaka, runtime kustom, atau file konfigurasi.

Bagian ini menjelaskan cara membuat dan menghapus layer di Lambda. Untuk informasi konseptual lebih lanjut tentang lapisan dan mengapa Anda mungkin mempertimbangkan untuk menggunakannya, lihatthe section called “Lapisan”.

Setelah Anda mengemas konten layer Anda, langkah selanjutnya adalah membuat layer di Lambda. Bagian ini menunjukkan cara membuat dan menghapus lapisan menggunakan konsol Lambda atau API Lambda saja. Untuk membuat layer menggunakan AWS CloudFormation, lihatthe section called “Lapisan dengan CloudFormation”. Untuk membuat layer menggunakan AWS Serverless Application Model (AWS SAM), lihatthe section called “Lapisan dengan AWS SAM”.

Topik

- Membuat lapisan
- Menghapus versi lapisan

Membuat lapisan

Untuk membuat layer, Anda dapat mengunggah arsip file.zip dari mesin lokal Anda atau dari Amazon Simple Storage Service (Amazon S3). Saat mengunggah dari Amazon S3, Anda dapat memilih apakah Lambda menyalin kode Lambda-managed ke penyimpanan atau mereferensikannya langsung dari bucket S3 Anda. Untuk informasi selengkapnya, lihat the section called “Self-managed Penyimpanan kode S3”.

Lambda mengekstrak konten layer ke dalam `/opt` direktori saat menyiapkan lingkungan eksekusi untuk Lambda Event Function.

Lapisan dapat memiliki satu atau lebih versi lapisan. Saat Anda membuat lapisan, Lambda menetapkan versi lapisan ke versi 1. Anda dapat mengubah izin pada versi lapisan yang ada kapan saja. Namun, untuk memperbarui kode atau membuat perubahan konfigurasi lainnya, Anda harus membuat versi baru dari layer.

Untuk membuat lapisan (konsol)

1. Buka halaman Fungsi di konsol Lambda.

2. Pilih Buat lapisan.
3. Di bagian Konfigurasi lapisan, untuk Nama, masukkan nama untuk lapisan Anda.
4. (Opsional) Untuk Deskripsi, masukkan deskripsi untuk lapisan Anda.
5. Untuk mengunggah kode lapisan Anda, lakukan salah satu hal berikut:
   - Untuk mengunggah file .zip dari komputer Anda, pilih Unggah file .zip. Selanjutnya, pilih Unggah untuk memilih file .zip lokal Anda.
   - Untuk mengunggah file dari Amazon S3, pilih Unggah file dari Amazon S3. Kemudian, untuk URL tautan Amazon S3, masukkan tautan ke file.
6. Jika Anda memilih untuk mengunggah dari Amazon S3, di bawah mode penyimpanan Kode, pilih salah satu dari berikut ini:
   - Mode salin (default)
   - Mode referensi
7. (Opsional) Untuk arsitektur yang kompatibel, pilih satu nilai atau kedua nilai. Untuk informasi selengkapnya, lihat the section called “Set instruksi (ARM/x86)”.
8. (Opsional) Untuk runtime yang kompatibel, pilih runtime yang kompatibel dengan layer Anda.
9. (Opsional) Untuk lisensi, masukkan informasi lisensi yang diperlukan.
10. Pilih Buat.

Atau, Anda dapat menjalankan perintah publish-layer-version AWS Command Line Interface (CLI). Saat mengunggah dari Amazon S3, `S3ObjectStorageMode=REFERENCE` sertakan untuk menggunakan penyimpanan kode S3 yang dikelola sendiri, atau hilangkan (default `COPY` ke) untuk menggunakan penyimpanan. Lambda-managed Contoh:

```
aws lambda publish-layer-version --layer-name my-layer --zip-file fileb://layer.zip --
compatible-runtimes python3.14
```

Setiap kali Anda menjalankan`publish-layer-version`, Lambda membuat versi baru dari layer.

Menghapus versi lapisan

Untuk menghapus versi layer, gunakan operasi DeleteLayerVersionAPI. Misalnya, jalankan AWS CLI perintah delete-layer-version dengan nama layer dan versi layer yang ditentukan.

```
aws lambda delete-layer-version --layer-name my-layer --version-number 1
```

Ketika Anda menghapus versi layer, Anda tidak dapat lagi mengkonfigurasi Fungsi Acara baru untuk menggunakannya. Namun, Fungsi Acara apa pun yang sudah mereferensikan versi lapisan yang dihapus terus berjalan dengan konten lapisan itu. Lambda tidak pernah menggunakan kembali nomor versi untuk nama lapisan.

Dampak kuota penyimpanan dari penghapusan lapisan

Dampak kuota penyimpanan dari menghapus versi lapisan tergantung pada mode penyimpanan yang digunakan oleh lapisan dan oleh fungsi yang mereferensikannya.

Lapisan menggunakan Lambda-managed penyimpanan (COPY)

Saat Anda menghapus versi lapisan yang menggunakan Lambda-managed penyimpanan, kode lapisan akan dihapus dari kuota Lambda-managed penyimpanan akun Anda. Namun, Fungsi Acara apa pun yang masih mereferensikan versi lapisan yang dihapus terus memiliki akses ke konten lapisan. Konten lapisan diperhitungkan dalam kuota ukuran paket penerapan Fungsi Acara (250 MB dibuka ritsleting) tetapi tidak lagi diperhitungkan dalam kuota penyimpanan tingkat akun. Lambda-managed

Lapisan menggunakan penyimpanan S3 yang dikelola sendiri (REFERENSI)

Lapisan yang menggunakan penyimpanan kode S3 yang dikelola sendiri tidak menggunakan Lambda-managed penyimpanan, jadi menghapusnya tidak berdampak pada kuota penyimpanan Anda Lambda-managed . Fungsi Peristiwa yang mereferensikan versi lapisan yang dikelola sendiri yang dihapus terus berjalan normal. Namun, jika Anda juga menghapus objek sumber dari bucket S3 Anda atau mencabut akses Lambda ke sana, Anda tidak dapat lagi memperbarui konfigurasi Fungsi Acara dengan lapisan itu.

> **Note**
>
> Menghapus versi lapisan tidak pernah menonaktifkan Fungsi Acara yang mereferensikannya. Fungsi Acara tetap `Active` terlepas dari apakah lapisan yang digunakan Lambda-managed atau penyimpanan S3 yang dikelola sendiri. `InactiveStatus` hanya berlaku ketika Lambda kehilangan akses ke kode sumber Fungsi Peristiwa sendiri yang disimpan dalam bucket S3 yang dikelola sendiri.

Menambahkan lapisan ke Fungsi Acara

Lapisan Lambda adalah arsip file.zip yang berisi kode atau data tambahan. Lapisan biasanya berisi dependensi pustaka, runtime kustom, atau file konfigurasi.

Bagian ini menjelaskan cara menambahkan layer ke Lambda Event Function. Untuk informasi konseptual lebih lanjut tentang lapisan dan mengapa Anda mungkin mempertimbangkan untuk menggunakannya, lihatthe section called “Lapisan”.

Sebelum Anda dapat mengkonfigurasi Fungsi Acara untuk menggunakan lapisan, Anda harus:

- Package konten layer Anda
- Buat layer di Lambda
- Pastikan Anda memiliki izin untuk memanggil GetLayerVersionAPI pada versi layer. Untuk Fungsi Acara di Anda Akun AWS, Anda harus memiliki izin ini dalam kebijakan pengguna Anda. Untuk menggunakan lapisan di akun lain, pemilik akun tersebut harus memberi akun Anda izin dalam kebijakan berbasis sumber daya. Sebagai contoh, lihat the section called “Akses lapisan untuk akun lain”.

Anda dapat menambahkan hingga lima lapisan ke Fungsi Acara. Total ukuran unzip dari Event Function dan semua layer tidak dapat melebihi kuota ukuran paket deployment unzip sebesar 250 MB. Untuk informasi selengkapnya, lihat Kuota Lambda.

Fungsi Acara Anda dapat terus menggunakan versi lapisan apa pun yang telah Anda tambahkan, bahkan setelah versi lapisan itu dihapus, atau setelah izin Anda untuk mengakses lapisan dicabut. Namun, Anda tidak dapat membuat Fungsi Acara baru yang menggunakan versi layer yang dihapus.

Untuk menambahkan layer ke Fungsi Acara

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih Fungsi Acara.
3. Gulir ke bawah ke bagian Layers, lalu pilih Add a layer.
4. Di bawah Pilih layer, pilih sumber layer:
    a. AWS layer: Pilih dari daftar ekstensi AWS-managed.
    b. Custom layers: Pilih layer yang dibuat di file Anda Akun AWS.
    c. Tentukan ARN: Untuk menggunakan lapisan dari yang berbeda Akun AWS, seperti ekstensi pihak ketiga, masukkan Nama Sumber Daya Amazon (ARN).

5. Pilih Tambahkan.

Urutan di mana Anda menambahkan lapisan adalah urutan di mana Lambda menggabungkan konten lapisan ke dalam lingkungan eksekusi. Anda dapat mengubah urutan penggabungan lapisan menggunakan konsol.

Untuk memperbarui urutan penggabungan lapisan untuk Fungsi Acara (konsol)

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih Fungsi Acara untuk dikonfigurasi.
3. Di bawah Layers, pilih Edit
4. Pilih salah satu layer.
5. Pilih Merge sebelumnya atau Merge nanti untuk menyesuaikan urutan layer.
6. Pilih Simpan.

Lapisan berversi. Isi dari setiap versi lapisan tidak dapat diubah. Pemilik layer dapat merilis versi layer baru untuk menyediakan konten yang diperbarui. Anda dapat menggunakan konsol untuk memperbarui versi lapisan yang dilampirkan ke Fungsi Acara Anda.

Untuk memperbarui versi layer untuk Event Function (konsol)

1. Buka halaman Fungsi di konsol Lambda.
2. Pilih layer yang ingin Anda perbarui versinya.
3. Pilih tab Fungsi menggunakan versi ini.
4. Pilih Fungsi Acara yang ingin Anda ubah, lalu pilih Edit.
5. Untuk versi Layer, pilih versi layer yang akan diubah.
6. Pilih fungsi Perbarui.

Anda tidak dapat memperbarui versi lapisan Fungsi Acara di seluruh AWS akun.

Menemukan informasi lapisan

Untuk menemukan layer di akun Anda yang kompatibel dengan runtime Event Function, gunakan ListLayersAPI. Misalnya, Anda dapat menggunakan perintah list-layers (AWS Command Line Interface CLI) berikut:

```
aws lambda list-layers --compatible-runtime python3.14
```

Anda akan melihat output yang serupa dengan yang berikut:

```
{
    "Layers": [
        {
            "LayerName": "my-layer",
            "LayerArn": "arn:aws:lambda:us-east-2:123456789012:layer:my-layer",
            "LatestMatchingVersion": {
                "LayerVersionArn": "arn:aws:lambda:us-east-2:123456789012:layer:my-
layer:2",
                "Version": 2,
                "Description": "My layer",
                "CreatedDate": "2025-04-15T00:37:46.592+0000",
                "CompatibleRuntimes": [
                    "python3.14"
                ]
            }
        }
    ]
}
```

Untuk membuat daftar semua lapisan di akun Anda, hilangkan `--compatible-runtime` opsi. Detail respons menunjukkan versi terbaru dari setiap lapisan.

Anda juga bisa mendapatkan versi terbaru dari layer menggunakan ListLayerVersionsAPI. Misalnya, Anda dapat menggunakan perintah `list-layer-versions` CLI berikut:

```
aws lambda list-layer-versions --layer-name my-layer
```

Anda akan melihat output yang serupa dengan yang berikut:

```
{
    "LayerVersions": [
        {
            "LayerVersionArn": "arn:aws:lambda:us-east-2:123456789012:layer:my-
layer:2",
            "Version": 2,
            "Description": "My layer",
            "CreatedDate": "2023-11-15T00:37:46.592+0000",
```

```
            "CompatibleRuntimes": [
                "java11"
            ]
        },
        {
            "LayerVersionArn": "arn:aws:lambda:us-east-2:123456789012:layer:my-
layer:1",
            "Version": 1,
            "Description": "My layer",
            "CreatedDate": "2023-11-15T00:27:46.592+0000",
            "CompatibleRuntimes": [
                "java11"
            ]
        }
    ]
}
```

Penggunaan AWS CloudFormation dengan lapisan

Anda dapat menggunakan CloudFormation untuk membuat layer dan mengaitkan layer dengan Lambda Event Function Anda. Contoh template berikut membuat layer bernama `my-lambda-layer` dan melampirkan layer ke Event Function menggunakan properti Layers.

Dalam contoh ini, template menentukan Amazon Resource Name (ARN) dari peran eksekusi IAM yang ada. Anda juga dapat membuat peran eksekusi baru dalam template menggunakan sumber daya CloudFormation AWS: :IAM: :Role.

Fungsi Acara Anda tidak memerlukan izin khusus untuk menggunakan lapisan.

```
---
Description: CloudFormation Template for Lambda Event Function with Lambda Layer
Resources:
  MyLambdaLayer:
    Type: AWS::Lambda::LayerVersion
    Properties:
      LayerName: my-lambda-layer
      Description: My Lambda Layer
      Content:
        S3Bucket: amzn-s3-demo-bucket
        S3Key: my-layer.zip
      CompatibleRuntimes:
        - python3.9
        - python3.10
        - python3.11

  MyLambdaFunction:
    Type: AWS::Lambda::Function
    Properties:
      FunctionName: my-lambda-function
      Runtime: python3.9
      Handler: index.handler
      Timeout: 10
      Role: arn:aws:iam::111122223333:role/my_lambda_role
      Layers:
        - !Ref MyLambdaLayer
```

Anda juga dapat menggunakan AWS Serverless Application Model untuk mengelola lapisan. Lihat informasi yang lebih lengkap di the section called “Lapisan dengan AWS SAM”.

Penggunaan AWS SAM dengan lapisan

Anda dapat menggunakan AWS Serverless Application Model (AWS SAM) untuk mengotomatiskan pembuatan lapisan dalam aplikasi Anda. Jenis `AWS::Serverless::LayerVersion` sumber daya membuat versi lapisan yang dapat Anda referensikan dari konfigurasi Fungsi Peristiwa Lambda Anda.

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: 'AWS::Serverless-2016-10-31'
Description: AWS SAM Template for Lambda Event Function with Lambda Layer

Resources:
  MyLambdaLayer:
    Type: AWS::Serverless::LayerVersion
    Properties:
      LayerName: my-lambda-layer
      Description: My Lambda Layer
      ContentUri: s3://amzn-s3-demo-bucket/my-layer.zip
      CompatibleRuntimes:
        - python3.9
        - python3.10
        - python3.11

  MyLambdaFunction:
    Type: AWS::Serverless::Function
    Properties:
      FunctionName: MyLambdaFunction
      Runtime: python3.9
      Handler: app.handler
      CodeUri: s3://amzn-s3-demo-bucket/my-function
      Layers:
        - !Ref MyLambdaLayer
```

Menambah Fungsi Acara Lambda menggunakan ekstensi Lambda

Anda dapat menggunakan ekstensi Lambda untuk menambah Fungsi Acara Lambda Anda. Misalnya, gunakan ekstensi Lambda untuk mengintegrasikan Fungsi Acara dengan alat pemantauan, observabilitas, keamanan, dan tata kelola pilihan Anda. Anda dapat memilih dari serangkaian luas alat yang disediakan oleh AWS Lambda Mitra, atau Anda dapat membuat ekstensi Lambda Anda sendiri.

Lambda mendukung ekstensi eksternal dan internal. Ekstensi eksternal berjalan sebagai proses independen di lingkungan eksekusi dan terus berjalan setelah pemanggilan Fungsi Acara diproses sepenuhnya. Karena ekstensi berjalan sebagai proses terpisah, Anda dapat menulisnya dalam bahasa yang berbeda dari Fungsi Acara. Semua ekstensi Runtime Fungsi Peristiwa Lambda dukungan.

Ekstensi internal berjalan sebagai bagian dari proses runtime. Fungsi Acara Anda mengakses ekstensi internal dengan menggunakan skrip pembungkus atau mekanisme dalam proses seperti. `JAVA_TOOL_OPTIONS` Untuk informasi selengkapnya, lihat Memodifikasi lingkungan waktu pengoperasian.

Anda dapat menambahkan ekstensi ke Fungsi Peristiwa menggunakan konsol Lambda, layanan AWS Command Line Interface (AWS CLI), atau infrastruktur sebagai kode (IAc) dan alat seperti CloudFormation, AWS Serverless Application Model (AWS SAM), dan Terraform.

Anda dikenakan biaya untuk waktu eksekusi yang digunakan oleh ekstensi (dengan penambahan 1 ms). Tidak ada biaya untuk menginstal ekstensi Anda sendiri. Untuk informasi harga ekstensi selengkapnya, lihat AWS Lambda Harga. Untuk informasi harga ekstensi mitra, lihat situs web mitra tersebut. Lihat the section called “Mitra ekstensi” daftar ekstensi mitra resmi.

Topik

- Lingkungan eksekusi
- Dampak pada performa dan sumber daya
- Izin
- Mengkonfigurasi ekstensi Lambda
- AWS Lambda mitra ekstensi
- Menggunakan API Ekstensi Lambda untuk membuat ekstensi
- Mengakses data telemetri real-time untuk ekstensi menggunakan API Telemetri

Lingkungan eksekusi

Lambda memanggil Fungsi Acara Anda di lingkungan eksekusi, yang menyediakan lingkungan runtime yang aman dan terisolasi. Lingkungan eksekusi mengelola sumber daya yang diperlukan untuk menjalankan Fungsi Acara Anda dan menyediakan dukungan siklus hidup untuk runtime dan ekstensi Fungsi Acara.

Siklus hidup lingkungan eksekusi mencakup fase-fase berikut:

- **`Init`**: Dalam fase ini, Lambda membuat atau membatalkan pembekuan lingkungan eksekusi dengan sumber daya terkonfigurasi, mengunduh kode untuk fungsi dan semua lapisan, menginisialisasi ekstensi apa pun, menginisialisasi runtime, lalu menjalankan kode inisialisasi fungsi (yaitu kode di luar handler utama). Fase `Init` terjadi saat invokasi pertama, atau sebelum invokasi fungsi jika Anda telah mengaktifkan konkurensi terprovisi.

  Fase `Init` dibagi ke dalam tiga subfase: `Extension init`, `Runtime init`, dan `Function init`. Subfase ini memastikan semua ekstensi dan runtime menyelesaikan tugas penyiapam mereka sebelum kode fungsi berjalan.

  Ketika SnapStart diaktifkan, `Init` fase terjadi ketika Anda menerbitkan versi fungsi. Lambda menyimpan snapshot memori dan status disk dari lingkungan eksekusi yang diinisialisasi, mempertahankan snapshot terenkripsi, dan menyimpannya dalam cache untuk akses latensi rendah. Jika Anda memiliki hook runtime sebelum-checkpoint, maka kode berjalan di akhir fase. `Init`
- **`Restore`**(SnapStart only): Saat Anda pertama kali memanggil SnapStartfungsi dan saat fungsi meningkat, Lambda melanjutkan lingkungan eksekusi baru dari snapshot yang bertahan alih-alih menginisialisasi fungsi dari awal. Jika Anda memiliki hook runtime after-restore, kode berjalan di akhir fase. `Restore` Anda dikenakan biaya untuk durasi kait runtime setelah pemulihan. Runtime harus dimuat dan kait runtime setelah pemulihan harus selesai dalam batas waktu tunggu (10 detik). Jika tidak, Anda akan mendapatkan SnapStartTimeoutException. Ketika `Restore` fase selesai, Lambda memanggil fungsi handler (the). Fase invokasi
- **`Invoke`**Dalam fase ini, Lambda memicu handler fungsi. Setelah fungsi berjalan hingga selesai, Lambda bersiap untuk menangani invokasi fungsi lain.
- **`Shutdown`**: Fase ini dipicu jika fungsi Lambda tidak menerima invokasi apa pun selama jangka waktu tertentu. Di fase `Shutdown`, Lambda menonaktifkan runtime, memberi tahu ekstensi agar mereka berhenti dengan bersih, lalu menghapus lingkungan. Lambda mengirim peristiwa `Shutdown` ke setiap ekstensi, yang memberi tahu ekstensi bahwa lingkungan akan dimatikan.

Selama fase `Init`, Lambda mengekstrak lapisan yang berisi ekstensi ke direktori `/opt` di lingkungan eksekusi. Lambda mencari ekstensi di direktori `/opt/extensions/`, menginterpretasikan setiap file sebagai bootstrap yang dapat dijalankan untuk meluncurkan ekstensi, dan memulai semua ekstensi secara paralel.

Dampak pada performa dan sumber daya

Ukuran ekstensi Fungsi Acara Anda diperhitungkan terhadap batas ukuran paket penerapan. Untuk arsip file.zip, ukuran total Fungsi Acara yang tidak di-zip dan semua ekstensi tidak dapat melebihi batas ukuran paket penerapan yang tidak di-zip sebesar 250 MB.

Ekstensi dapat memengaruhi kinerja Fungsi Acara Anda karena mereka berbagi sumber daya Fungsi Acara seperti CPU, memori, dan penyimpanan. Misalnya, jika ekstensi melakukan operasi komputasi intensif, Anda mungkin melihat durasi eksekusi Event Function Anda meningkat.

Setiap ekstensi harus menyelesaikan inisialisasi sebelum Lambda memanggil Fungsi Acara. Oleh karena itu, ekstensi yang menghabiskan waktu inisialisasi yang signifikan dapat meningkatkan latensi pemanggilan Fungsi Peristiwa.

Untuk mengukur waktu tambahan yang dibutuhkan ekstensi setelah eksekusi Fungsi Acara, Anda dapat menggunakan metrik `PostRuntimeExtensionsDuration` fungsi. Untuk mengukur peningkatan memori yang digunakan, Anda dapat menggunakan metrik `MaxMemoryUsed`. Untuk memahami dampak ekstensi tertentu, Anda dapat menjalankan berbagai versi Fungsi Acara Anda secara berdampingan.

> **Note**
>
> `MaxMemoryUsed`metrik adalah salah satu Metrik yang dikumpulkan oleh Lambda Insights dan bukan metrik asli Lambda.

Izin

Ekstensi memiliki akses ke sumber daya yang sama dengan Fungsi Acara. Karena ekstensi dijalankan dalam lingkungan yang sama dengan Fungsi Acara, izin dibagi antara Fungsi Acara dan ekstensi.

Untuk arsip file.zip, Anda dapat membuat CloudFormation templat untuk menyederhanakan tugas melampirkan konfigurasi ekstensi yang sama—termasuk izin AWS Identity and Access Management (IAM) — ke beberapa Fungsi Acara.

Mengkonfigurasi ekstensi Lambda

Mengonfigurasi ekstensi (arsip file .zip)

Anda dapat menambahkan ekstensi ke Fungsi Acara Lambda Anda sebagai lapisan Lambda. Menggunakan lapisan, Anda dapat berbagi ekstensi di seluruh organisasi Anda atau ke seluruh komunitas pengembang Lambda. Anda dapat menambahkan satu ekstensi atau lebih ke satu lapisan. Anda dapat mendaftarkan hingga 10 ekstensi untuk Fungsi Acara.

Anda menambahkan ekstensi ke Fungsi Acara Anda menggunakan metode yang sama seperti yang Anda lakukan untuk lapisan apa pun. Untuk informasi selengkapnya, lihat the section called "Lapisan".

Tambahkan ekstensi ke Fungsi Acara Anda (konsol)

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih Fungsi Acara.
3. Pilih tab Kode jika belum dipilih.
4. Di bagian Lapisan, pilih Edit.
5. Untuk Pilih lapisan, pilih Tentukan ARN.
6. Untuk Tentukan ARN, masukkan Amazon Resource Name (ARN) dari lapisan ekstensi.
7. Pilih Tambahkan.

Menggunakan ekstensi dalam gambar kontainer

Anda dapat menambahkan ekstensi ke gambar kontainer Anda. Pengaturan gambar kontainer ENTRYPOINT menentukan proses utama untuk Fungsi Acara. Konfigurasikan pengaturan ENTRYPOINT di Dockerfile, atau sebagai penggantian dalam konfigurasi Fungsi Acara.

Anda dapat menjalankan beberapa proses dalam satu kontainer. Lambda mengelola siklus hidup proses utama dan proses tambahan. Lambda menggunakanAPI Ekstensiuntuk mengelola siklus hidup ekstensi.

Contoh: Menambahkan ekstensi eksternal

Ekstensi eksternal berjalan dalam proses terpisah dari Fungsi Peristiwa Lambda. Lambda memulai proses untuk setiap ekstensi di direktori `/opt/extensions/`. Lambda menggunakan API Ekstensi

untuk mengelola siklus hidup ekstensi. Setelah Fungsi Acara berjalan hingga selesai, Lambda mengirimkan `Shutdown` acara ke setiap ekstensi eksternal.

Example dari menambahkan ekstensi eksternal ke gambar dasar Python

```
FROM public.ecr.aws/lambda/python:3.11

# Copy and install the app
COPY /app /app
WORKDIR /app
RUN pip install -r requirements.txt

# Add an extension from the local directory into /opt/extensions
ADD my-extension.zip /opt/extensions
CMD python ./my-function.py
```

Langkah selanjutnya

Untuk mempelajari selengkapnya tentang ekstensi, kami merekomendasikan sumber daya berikut:

- Untuk contoh kerja dasar, lihat Membangun Ekstensi untuk AWS Lambda di AWS Compute Blog.
- Untuk informasi tentang ekstensi yang disediakan AWS Lambda Partner, lihat Memperkenalkan AWS Lambda Ekstensi di Blog AWS Komputasi.
- Untuk melihat contoh ekstensi dan skrip pembungkus yang tersedia, lihat AWS Lambda Ekstensi pada repositori AWS Sampel GitHub .

AWS Lambda mitra ekstensi

AWS Lambda telah bermitra dengan beberapa entitas pihak ketiga untuk menyediakan ekstensi untuk diintegrasikan dengan Fungsi Acara Lambda Anda. Daftar berikut merinci ekstensi pihak ketiga yang siap Anda gunakan kapan saja.

- AppDynamics— Menyediakan instrumentasi otomatis Node.js atau Fungsi Peristiwa Lambda Python, memberikan visibilitas dan peringatan pada kinerja Fungsi Acara.
- Axiom— Menyediakan dasbor untuk memantau kinerja Fungsi Peristiwa Lambda dan metrik tingkat sistem agregat.
- Datadog— Menyediakan visibilitas real-time yang komprehensif ke aplikasi tanpa server Anda melalui penggunaan metrik, jejak, dan log.
- Dynatrace— Memberikan visibilitas ke dalam jejak dan metrik, dan menggunakan AI untuk deteksi kesalahan otomatis dan analisis akar penyebab di seluruh tumpukan aplikasi.
- ElasticMenyediakan Application Performance Monitoring (APM) untuk mengidentifikasi dan menyelesaikan masalah akar penyebab menggunakan jejak, metrik, dan log yang berkorelasi.
- Epsagon— Mendengarkan acara pemanggilan, menyimpan jejak, dan mengirimkannya secara paralel dengan eksekusi Fungsi Peristiwa Lambda.
- Fastly— Melindungi Fungsi Acara Lambda Anda dari aktivitas mencurigakan, seperti serangan gaya injeksi, pengambilalihan akun melalui isian kredenal, bot berbahaya, dan penyalahgunaan API.
- HashiCorp Vault— Mengelola rahasia dan membuatnya tersedia bagi pengembang untuk digunakan dalam kode Fungsi Acara, tanpa membuat Event Functions Vault sadar.
- Honeycomb— Alat observabilitas untuk men-debug tumpukan aplikasi Anda.
- Lumigo— Profil Lambda Event Function memanggil dan mengumpulkan metrik untuk memecahkan masalah di lingkungan tanpa server dan layanan mikro.
- New Relic— Berjalan bersama Fungsi Acara Lambda, secara otomatis mengumpulkan, meningkatkan, dan mengangkut telemetri ke platform observabilitas terpadu New Relic.
- Sedai— Platform manajemen cloud otonom, didukung oleh AI/ML, yang memberikan pengoptimalan berkelanjutan untuk tim operasi cloud untuk memaksimalkan penghematan biaya cloud, kinerja, dan ketersediaan dalam skala besar.
- Sentry— Mendiagnosis, memperbaiki, dan mengoptimalkan kinerja Fungsi Peristiwa Lambda.
- Site24x7— Mencapai observabilitas waktu nyata ke lingkungan Lambda Anda

- Splunk— Mengumpulkan metrik resolusi tinggi, latensi rendah untuk pemantauan Fungsi Peristiwa Lambda yang efisien dan efektif.
- Sumo Logic— Memberikan visibilitas ke dalam kesehatan dan kinerja aplikasi tanpa server.
- Salt Security - Menyederhanakan tata kelola postur API dan keamanan API untuk Fungsi Acara Lambda melalui pengaturan otomatis dan dukungan untuk runtime yang beragam.

AWS ekstensi terkelola

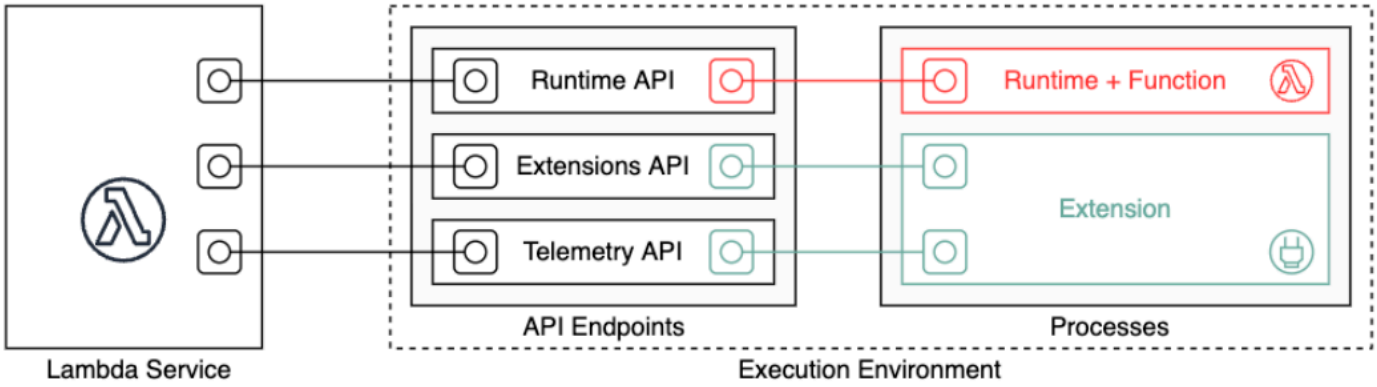
AWS menyediakan ekstensi terkelola sendiri, termasuk:

- AWS AppConfig— Gunakan bendera fitur dan data dinamis untuk memperbarui Fungsi Acara Lambda Anda. Anda juga dapat menggunakan ekstensi ini untuk memperbarui konfigurasi dinamis lainnya, seperti pelambatan operasi dan penyetelan.
- Amazon CodeGuru Profiler — Meningkatkan kinerja aplikasi dan mengurangi biaya dengan menentukan baris kode aplikasi yang paling mahal dan memberikan rekomendasi untuk meningkatkan kode.
- CloudWatch Lambda Insights — Pantau, pecahkan masalah, dan optimalkan kinerja Fungsi Acara Lambda Anda melalui dasbor otomatis.
- AWS Distro for OpenTelemetry (ADOT) - Memungkinkan Fungsi Acara untuk mengirim data jejak ke layanan AWS pemantauan seperti AWS X-Ray, dan ke tujuan yang mendukung OpenTelemetry seperti Honeycomb dan Lightstep.
- AWS Parameter dan Rahasia - Mengambil parameter dengan aman dari AWS Systems Manager Parameter Store dan rahasia dari dalam Fungsi Acara AWS Secrets Manager Lambda.

Untuk contoh ekstensi tambahan dan proyek demo, lihat AWS Lambda Ekstensi.

Menggunakan API Ekstensi Lambda untuk membuat ekstensi

Lambda Event Function penulis menggunakan ekstensi untuk mengintegrasikan Lambda dengan alat pilihan mereka untuk pemantauan, observabilitas, keamanan, dan tata kelola. Penulis Fungsi Acara dapat menggunakan ekstensi dari AWS, AWS Mitra, dan proyek sumber terbuka. Untuk informasi selengkapnya tentang penggunaan ekstensi, lihat Memperkenalkan AWS Lambda Ekstensi di Blog AWS Komputasi. Bagian ini menjelaskan cara menggunakan API Ekstensi Lambda, siklus hidup lingkungan eksekusi Lambda, dan referensi API Ekstensi Lambda.



Sebagai penulis ekstensi, Anda dapat menggunakan API Ekstensi Lambda untuk mengintegrasikan secara mendalam ke Lingkungan eksekusi Lambda. Ekstensi Anda dapat mendaftar untuk acara siklus hidup Event Function dan lingkungan eksekusi. Menanggapi peristiwa ini, Anda dapat memulai proses baru, menjalankan logika, dan mengontrol serta berpartisipasi dalam semua fase siklus hidup Lambda: inisialisasi, invokasi, dan penonaktifan. Selain itu, Anda dapat menggunakan API Log Runtime untuk menerima aliran log.

Ekstensi berjalan sebagai proses independen di lingkungan eksekusi dan dapat terus berjalan setelah pemanggilan Fungsi Acara diproses sepenuhnya. Karena ekstensi berjalan sebagai proses, Anda dapat menulisnya dalam bahasa yang berbeda dari Fungsi Acara. Sebaiknya terapkan ekstensi eksternal menggunakan bahasa terkompilasi.

Dalam hal ini, ekstensi adalah biner mandiri yang kompatibel dengan runtime yang didukung. Semua ekstensi Runtime Fungsi Peristiwa Lambda dukungan. Jika Anda menggunakan bahasa nonkompilasi, pastikan Anda memasukkan waktu pengoperasian yang kompatibel di ekstensi.

Lambda juga mendukung ekstensi internal. Ekstensi internal berjalan sebagai thread terpisah dari proses runtime. Runtime memulai dan menghentikan ekstensi internal. Cara alternatif untuk berintegrasi dengan lingkungan Lambda adalah dengan menggunakan variabel lingkungan dan skrip

wrapper khusus bahasa. Anda dapat menggunakan ini untuk mengonfigurasi lingkungan runtime dan memodifikasi perilaku startup proses runtime.

Anda dapat menambahkan ekstensi ke Fungsi Acara dengan dua cara. Untuk Fungsi Peristiwa yang digunakan sebagai arsip file.zip, Anda menerapkan ekstensi Anda sebagai lapisan. Untuk Fungsi Peristiwa yang didefinisikan sebagai gambar kontainer, Anda menambahkan ekstensi ke gambar kontainer Anda.

> **Note**
>
> Misalnya ekstensi dan skrip pembungkus, lihat AWS Lambda Ekstensi pada repositori AWS Sampel GitHub .

Topik

- Siklus hidup lingkungan eksekusi Lambda
- Referensi API ekstensi

Siklus hidup lingkungan eksekusi Lambda

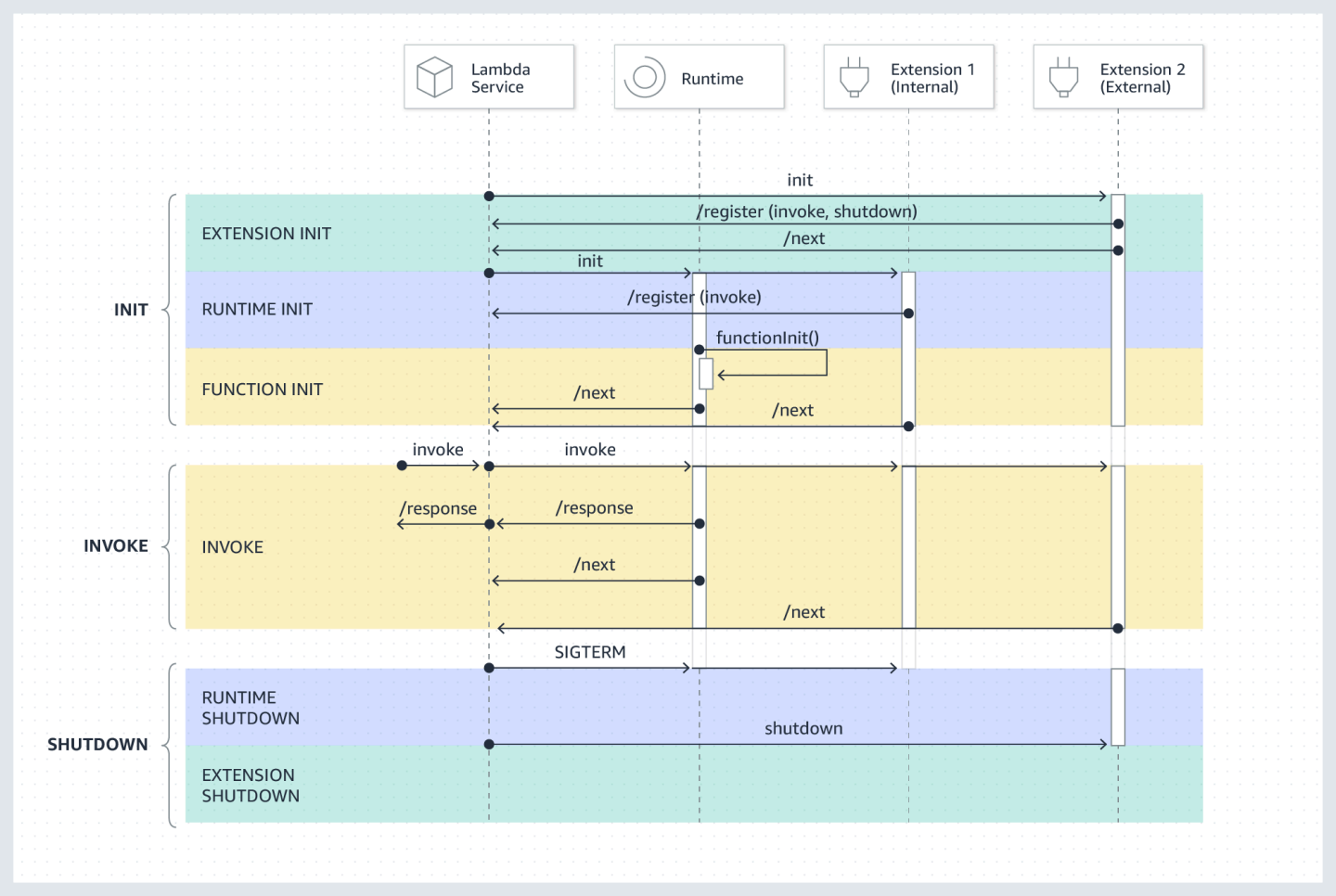
Siklus hidup lingkungan eksekusi mencakup fase-fase berikut:

- **`Init`**: Pada fase ini, Lambda membuat atau tidak membekukan lingkungan eksekusi dengan sumber daya yang dikonfigurasi, mengunduh kode untuk Fungsi Acara dan semua lapisan, menginisialisasi ekstensi apa pun, menginisialisasi runtime, dan kemudian menjalankan kode inisialisasi Fungsi Acara (kode di luar pengendali utama). `Init`Fase terjadi baik selama pemanggilan pertama, atau sebelum pemanggilan Fungsi Acara jika Anda telah mengaktifkan konkurensi yang disediakan.

  Fase `Init` dibagi ke dalam tiga subfase: `Extension init`, `Runtime init`, dan `Function init`. Sub-fase ini memastikan bahwa semua ekstensi dan runtime menyelesaikan tugas penyiapannya sebelum kode Fungsi Acara berjalan.
- **`Invoke`**: Pada fase ini, Lambda memanggil handler Event Function. Setelah Fungsi Acara berjalan hingga selesai, Lambda bersiap untuk menangani pemanggilan Fungsi Acara lainnya.
- **`Shutdown`**: Fase ini dipicu jika Fungsi Peristiwa Lambda tidak menerima pemanggilan apa pun untuk jangka waktu tertentu. Di fase `Shutdown`, Lambda menonaktifkan runtime, memberi tahu ekstensi agar mereka berhenti dengan bersih, lalu menghapus lingkungan. Lambda mengirim

kejadian `Shutdown` ke setiap ekstensi, yang memberi tahu ekstensi bahwa lingkungan akan dimatikan.

Setiap fase dimulai dengan peristiwa dari Lambda ke runtime dan ke semua ekstensi terdaftar. Waktu pengoperasian dan setiap ekstensi memberi sinyal penyelesaian dengan mengirimkan permintaan API `Next`. Lambda membekukan lingkungan eksekusi saat waktu proses telah selesai dan tidak ada kejadian yang tertunda.



Topik

- Fase inisialisasi
- Fase invokasi
- Fase pematian
- Izin dan konfigurasi
- Penanganan kegagalan

- Pemecahan masalah ekstensi

Fase inisialisasi

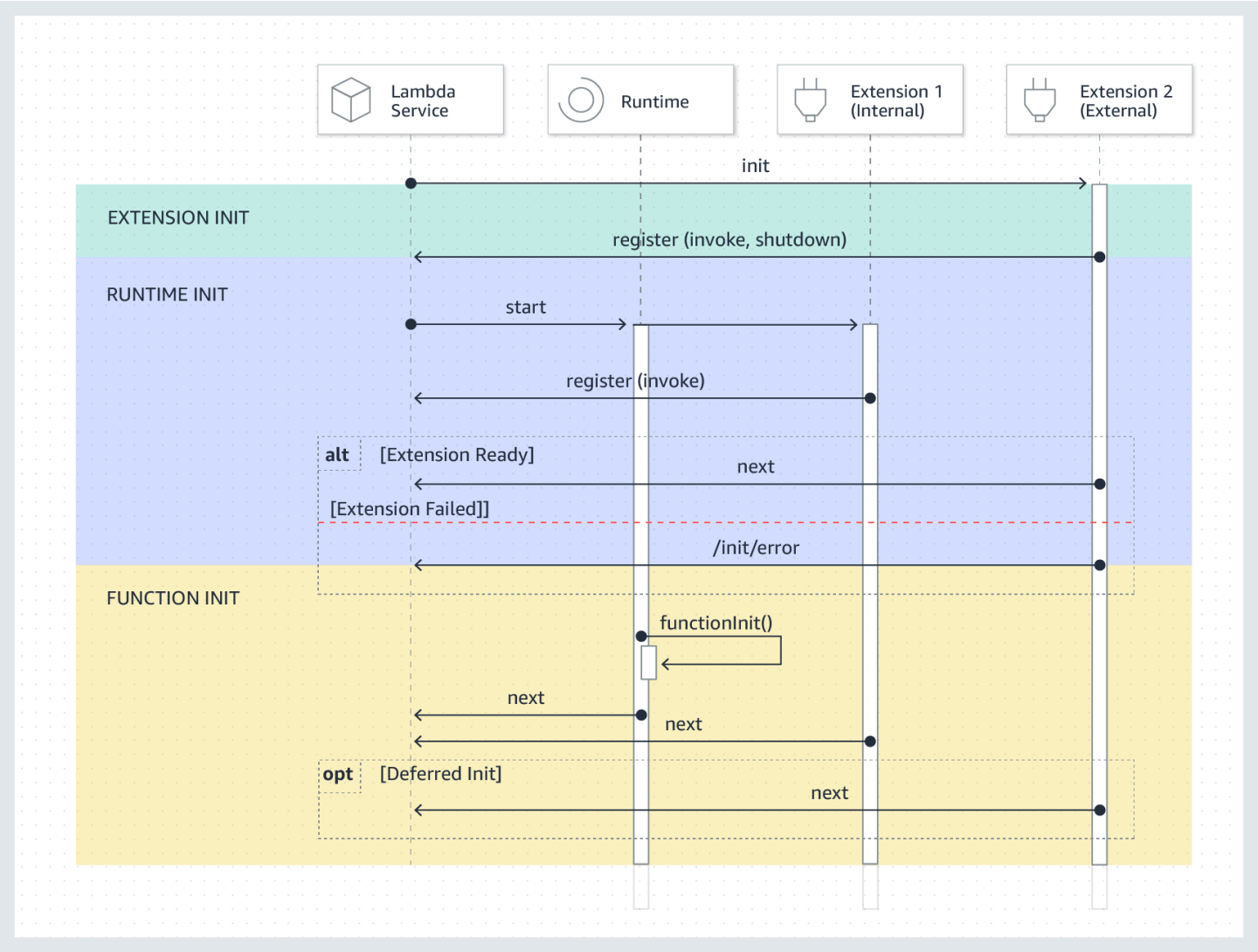
Selama fase `Extension init`, setiap ekstensi perlu mendaftarkan dengan Lambda untuk menerima kejadian. Lambda menggunakan nama file ekstensi lengkap untuk memvalidasi bahwa ekstensi telah menyelesaikan sekuens bootstrap. Oleh karena itu, tiap panggilan API `Register` harus menyertakan header `Lambda-Extension-Name` dengan nama ekstensi file lengkap.

Anda dapat mendaftarkan hingga 10 ekstensi untuk Fungsi Acara. Batasan ini ditegakkan melalui panggilan API `Register`.

Setelah setiap ekstensi mendaftar, Lambda memulai fase `Runtime init`. Proses runtime memanggil `functionInit` untuk memulai fase `Function init`.

Fase `Init` selesai setelah waktu pengoperasian dan setiap ekstensi terdaftar menunjukkan penyelesaian dengan mengirim permintaan API `Next`.

> **Note**
>
> Ekstensi dapat menyelesaikan inisialisasinya dalam fase `Init` kapan pun.



Fase invokasi

Saat Fungsi Peristiwa Lambda dipanggil sebagai respons terhadap permintaan `Next` API, Lambda mengirimkan `Invoke` peristiwa ke runtime dan ke setiap ekstensi yang terdaftar untuk acara tersebut. `Invoke`

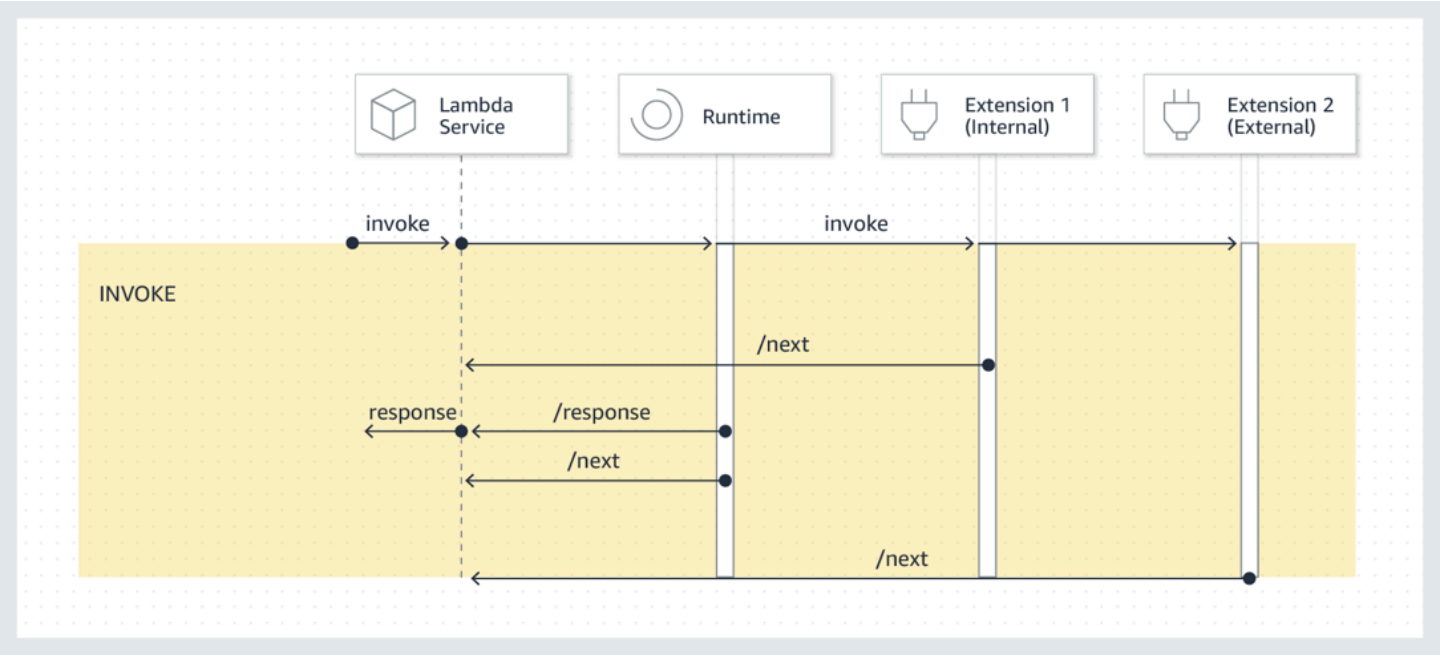
> **Note**
>
> Instans Terkelola Lambda: Ekstensi untuk fungsi Instans Terkelola Lambda tidak dapat mendaftar untuk acara tersebut. `Invoke` Karena Instans Terkelola Lambda mendukung pemanggilan bersamaan dalam satu lingkungan eksekusi, acara tidak didukung. `Invoke` Ekstensi hanya dapat mendaftar untuk `Shutdown` acara tersebut. Jika Anda perlu melacak

kapan pemanggilan dimulai dan selesai, gunakan acara `platform.report` platform melalui API Telemetri.

Selama pemanggilan, ekstensi eksternal berjalan secara paralel dengan Fungsi Acara. Mereka juga terus berjalan setelah Fungsi Acara selesai. Anda dapat menggunakan ini untuk menangkap informasi diagnostik atau mengirim log, metrik, dan jejak ke lokasi pilihan Anda.

Setelah menerima respons Fungsi Peristiwa dari runtime, Lambda mengembalikan respons ke klien, meskipun ekstensi masih berjalan.

Fase `Invoke` berakhir ketika waktu pengoperasian dan semua ekstensi menandakan bahwa ekstensi selesai dengan mengirim permintaan API `Next`.



Payload peristiwa: Acara yang dikirim ke runtime (dan Fungsi Peristiwa Lambda) membawa seluruh permintaan, header (sepertiRequestId), dan payload. Kejadian yang dikirim ke setiap ekstensi berisi metadata yang menjelaskan konten kejadian. Peristiwa siklus hidup ini mencakup jenis peristiwa, waktu waktu fungsi peristiwa habis (`deadlineMs`), Nama Sumber Daya Amazon (ARN) Fungsi Peristiwa yang dipanggil`requestId`, dan header penelusuran.

Ekstensi yang ingin mengakses badan peristiwa Fungsi Acara dapat menggunakan SDK dalam proses yang berkomunikasi dengan ekstensi. Pengembang Fungsi Acara menggunakan SDK dalam proses untuk mengirim payload ke ekstensi saat Fungsi Peristiwa dipanggil.

Berikut adalah contoh muatan:

```
{
    "eventType": "INVOKE",
    "deadlineMs": 676051,
    "requestId": "3da1f2dc-3222-475e-9205-e2e6c6318895",
    "invokedFunctionArn": "arn:aws:lambda:us-
east-1:123456789012:function:ExtensionTest",
    "tracing": {
        "type": "X-Amzn-Trace-Id",
        "value":
 "Root=1-5f35ae12-0c0fec141ab77a00bc047aa2;Parent=2be948a625588e32;Sampled=1"
    }
 }
```

Batas durasi: Pengaturan batas waktu Fungsi Acara membatasi durasi seluruh `Invoke` fase. Misalnya, jika Anda mengatur batas waktu Fungsi Acara sebagai 360 detik, Fungsi Acara dan semua ekstensi harus diselesaikan dalam 360 detik. Perhatikan bahwa tidak ada fase pasca-invokasi yang independen. Durasi adalah total waktu yang dibutuhkan untuk runtime Anda dan semua pemanggilan ekstensi Anda selesai dan tidak dihitung sampai Fungsi Acara dan semua ekstensi selesai berjalan.

Dampak kinerja dan overhead ekstensi: Ekstensi dapat memengaruhi kinerja Fungsi Acara. Sebagai penulis ekstensi, Anda memiliki kontrol atas dampak kinerja dari ekstensi Anda. Misalnya, jika ekstensi Anda melakukan operasi komputasi intensif, durasi Event Function akan meningkat karena ekstensi dan kode Event Function berbagi sumber daya CPU yang sama. Selain itu, jika ekstensi Anda melakukan operasi ekstensif setelah pemanggilan Fungsi Acara selesai, durasi Fungsi Peristiwa meningkat karena `Invoke` fase berlanjut hingga semua ekstensi memberi sinyal bahwa ekstensi selesai.

> **Note**
>
> Lambda mengalokasikan daya CPU secara proporsional dengan pengaturan memori Event Function. Anda mungkin melihat peningkatan durasi eksekusi dan inisialisasi pada pengaturan memori yang lebih rendah karena Fungsi Acara dan proses ekstensi bersaing untuk sumber daya CPU yang sama. Untuk mengurangi eksekusi dan inisialisasi durasi, coba tingkatkan pengaturan memori.

Untuk membantu mengidentifikasi dampak performa yang muncul karena ekstensi di fase `Invoke`, Lambda mengeluarkan metrik `PostRuntimeExtensionsDuration`. Metrik ini mengukur waktu kumulatif yang dihabiskan antara waktu pengoperasian permintaan API `Next` dan ekstensi

permintaan API `Next` terakhir. Untuk mengukur peningkatan memori yang digunakan, gunakan metrik `MaxMemoryUsed`. Untuk informasi selengkapnya tentang metrik Fungsi Peristiwa, lihatMenggunakan CloudWatch metrik dengan Lambda.

Pengembang Fungsi Acara dapat menjalankan berbagai versi Fungsi Acara mereka secara berdampingan untuk memahami dampak ekstensi tertentu. Kami menyarankan agar pembuat ekstensi mempublikasikan konsumsi sumber daya yang diharapkan untuk memudahkan pengembang Fungsi Acara memilih ekstensi yang sesuai.

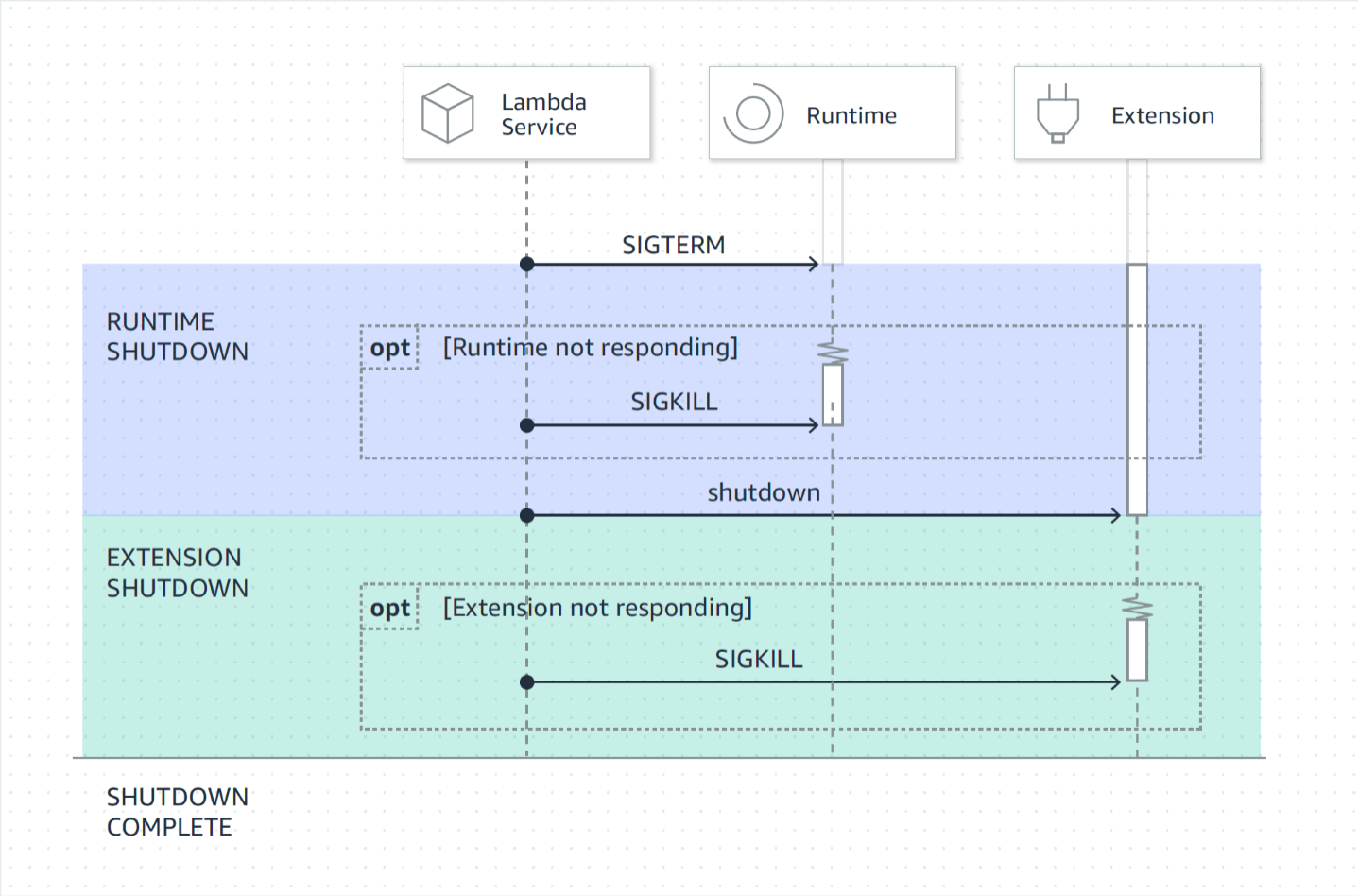
Fase pematian

Ketika Lambda akan mematikan runtime, Lambda mengirimkan `Shutdown` ke setiap ekstensi eksternal terdaftar. Ekstensi dapat menggunakan waktu ini untuk tugas pembersihan akhir. Kejadian `Shutdown` dikirim untuk menanggapi permintaan API `Next`.

Batas durasi: Durasi maksimum fase `Shutdown` bergantung pada konfigurasi ekstensi terdaftar:

- 0 ms - Fungsi tanpa ekstensi terdaftar
- 500 md – Fungsi dengan ekstensi internal terdaftar.
- 2.000 md – Fungsi dengan satu atau beberapa ekstensi eksternal terdaftar.

Jika waktu pengoperasian atau ekstensi tidak merespons kejadian `Shutdown` dalam batasan, Lambda mengakhiri proses menggunakan sinyal `SIGKILL`.



Muatan peristiwa: Peristiwa `Shutdown` tersebut berisi alasan penonaktifan dan waktu yang tersisa dalam milidetik.

`shutdownReason` mencakup nilai-nilai berikut:

- SPINDOWN – Pematian normal
- TIMEOUT – Batas waktu durasi habis
- FAILURE – Kondisi kesalahan, seperti kejadian `out-of-memory`

```
{
  "eventType": "SHUTDOWN",
  "shutdownReason": "reason for shutdown",
  "deadlineMs": "the time and date that the function times out in Unix time
 milliseconds"
}
```

Izin dan konfigurasi

Ekstensi berjalan di lingkungan eksekusi yang sama dengan Fungsi Peristiwa Lambda. Ekstensi juga berbagi sumber daya dengan Fungsi Acara, seperti CPU, memori, dan penyimpanan `/tmp` disk. Selain itu, ekstensi menggunakan peran AWS Identity and Access Management (IAM) dan konteks keamanan yang sama dengan Fungsi Acara.

Izin akses sistem file dan jaringan: Ekstensi berjalan di sistem file dan namespace nama jaringan yang sama dengan runtime Fungsi Acara. Ini berarti ekstensi harus kompatibel dengan sistem operasi terkait. Jika ekstensi memerlukan aturan lalu lintas jaringan keluar tambahan, Anda harus menerapkan aturan ini ke konfigurasi Fungsi Acara.

> **Note**
>
> Karena direktori kode Event Function hanya-baca, ekstensi tidak dapat mengubah kode Fungsi Acara.

Variabel lingkungan: Ekstensi dapat mengakses variabel lingkungan Fungsi Acara, kecuali untuk variabel berikut yang spesifik untuk proses runtime:

- `AWS_EXECUTION_ENV`
- `AWS_LAMBDA_LOG_GROUP_NAME`
- `AWS_LAMBDA_LOG_STREAM_NAME`
- `AWS_XRAY_CONTEXT_MISSING`
- `AWS_XRAY_DAEMON_ADDRESS`
- `LAMBDA_RUNTIME_DIR`
- `LAMBDA_TASK_ROOT`
- `_AWS_XRAY_DAEMON_ADDRESS`
- `_AWS_XRAY_DAEMON_PORT`
- `_HANDLER`

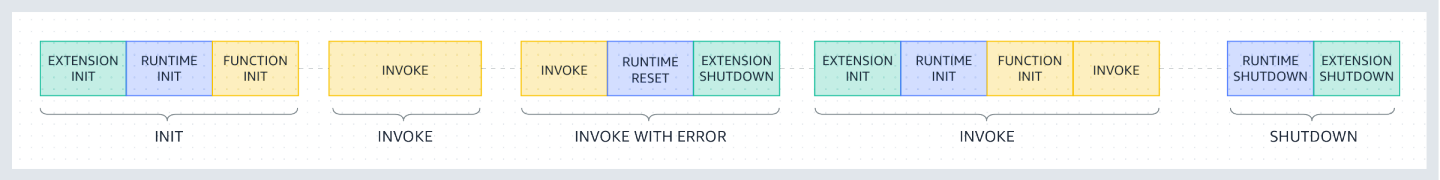
> **Note**
>
> Mendeteksi Instans Terkelola Lambda: Ekstensi dapat memeriksa `AWS_LAMBDA_INITIALIZATION_TYPE` variabel lingkungan untuk menentukan apakah mereka berjalan pada Instans Terkelola Lambda versus fungsi Lambda (default). Ini adalah metode yang direkomendasikan untuk ekstensi untuk menyesuaikan perilakunya berdasarkan jenis lingkungan eksekusi.

Penanganan kegagalan

Kegagalan insialisasi: Jika ekstensi gagal, Lambda memulai ulang lingkungan eksekusi untuk memberlakukan perilaku yang konsisten dan mendorong kegagalan cepat untuk ekstensi. Selain itu, untuk beberapa pelanggan, ekstensi tersebut harus memenuhi kebutuhan kritis misi, seperti pencatatan log, keamanan, tata kelola, dan pengumpulan telemetri.

Kegagalan pemanggilan (seperti kehabisan memori, batas waktu Fungsi Acara): Karena ekstensi berbagi sumber daya dengan runtime, kelelahan memori memengaruhi mereka. Saat waktu pengoperasian gagal, semua ekstensi dan waktu pengoperasian sendiri ikut serta dalam fase `Shutdown`. Selain itu, runtime dimulai ulang—baik secara otomatis sebagai bagian dari pemanggilan saat ini, atau melalui mekanisme inisialisasi ulang yang ditangguhkan.

Jika terjadi kegagalan (seperti batas waktu Fungsi Peristiwa atau kesalahan runtime) selama`Invoke`, layanan Lambda akan melakukan reset. Reset berperilaku seperti kejadian `Shutdown`. Pertama, Lambda mematikan waktu pengoperasian, lalu ia mengirim kejadian `Shutdown` untuk setiap ekstensi eksternal terdaftar. Kejadian ini mencakup alasan untuk pematian. Jika lingkungan ini digunakan untuk invokasi baru, ekstensi dan waktu pengoperasian diinisialisasi ulang sebagai bagian dari invokasi berikutnya.



Untuk penjelasan lebih rinci tentang diagram sebelumnya, lihat[Kegagalan selama fase pemanggilan](#).

Log ekstensi: Lambda mengirimkan keluaran log ekstensi ke CloudWatch Log. Lambda juga menghasilkan log acara tambahan untuk setiap ekstensi selama `Init`. Log acara mencatat preferensi nama dan registrasi (peristiwa, konfigurasi) saat berhasil, atau alasan kegagalan saat gagal.

Pemecahan masalah ekstensi

- Jika permintaan `Register` gagal, pastikan header `Lambda-Extension-Name` di panggilan API `Register` berisi nama file lengkap ekstensi.
- Jika permintaan `Register` gagal untuk ekstensi internal, pastikan permintaan tersebut tidak terdaftar untuk kejadian `Shutdown`.

Referensi API ekstensi

Spesifikasi OpenAPI untuk versi API ekstensi 2020-01-01 tersedia di sini: extensions-api.zip

Anda dapat mengambil nilai titik akhir API dari variabel lingkungan `AWS_LAMBDA_RUNTIME_API`. Untuk mengirim permintaan `Register`, gunakan prefiks `2020-01-01/` sebelum setiap jalur API. Sebagai contoh:

```
http://${AWS_LAMBDA_RUNTIME_API}/2020-01-01/extension/register
```

Metode API

- Pendaftaran
- Selanjutnya
- Kesalahan init
- Kesalahan keluar

Pendaftaran

Selama `Extension init`, semua ekstensi perlu mendaftar dengan Lambda untuk menerima kejadian. Lambda menggunakan nama file ekstensi lengkap untuk memvalidasi bahwa ekstensi telah menyelesaikan sekuens bootstrap. Oleh karena itu, tiap panggilan API `Register` harus menyertakan header `Lambda-Extension-Name` dengan nama ekstensi file lengkap.

Ekstensi internal dimulai dan dihentikan oleh proses waktu pengoperasian sehingga tidak diizinkan untuk terdaftar untuk kejadian `Shutdown`.

Jalur – `/extension/register`

Metode – POST

Minta header

- `Lambda-Extension-Name` – Nama file lengkap ekstensi. Wajib: ya. Jenis: string.
- `Lambda-Extension-Accept-Feature`— Gunakan ini untuk menentukan fitur Ekstensi opsional selama pendaftaran. Wajib: tidak. Jenis: string dipisahkan koma. Fitur yang tersedia untuk ditentukan menggunakan pengaturan ini:
  - `accountId`— Jika ditentukan, respons pendaftaran Ekstensi berisi ID akun yang terkait dengan Fungsi Peristiwa Lambda tempat Anda mendaftarkan Ekstensi.

Minta parameter tubuh

- `events` – Array kejadian yang akan didaftarkan. Wajib: tidak. Jenis: array string. String yang valid: `INVOKE`, `SHUTDOWN`.

> **Note**
>
> Instans Terkelola Lambda: Ekstensi untuk fungsi Instans Terkelola Lambda hanya dapat mendaftar untuk acara tersebut. `SHUTDOWN` Mencoba mendaftar untuk `INVOKE` acara tersebut menghasilkan kesalahan. Ini karena Instans Terkelola Lambda mendukung pemanggilan bersamaan dalam satu lingkungan eksekusi.

Header respons

- `Lambda-Extension-Identifier` – Menghasilkan pengidentifikasi agen unik (string UUID) yang diperlukan untuk semua permintaan berikutnya.

Kode respons

- 200 – Isi respons berisi nama fungsi, versi fungsi, dan nama penangan.
- 400 – Permintaan Buruk
- 403 – Dilarang
- 500 - Kesalahan kontainer. Non-recoverable negara. Ekstensi harus segera keluar.

Example Contoh isi permintaan

```
{
    'events': [ 'INVOKE', 'SHUTDOWN']
```

```
}
```

Example Contoh isi respons

```
{
    "functionName": "helloWorld",
    "functionVersion": "$LATEST",
    "handler": "lambda_function.lambda_handler"
}
```

Example Contoh badan respons dengan fitur AccountID opsional

```
{
    "functionName": "helloWorld",
    "functionVersion": "$LATEST",
    "handler": "lambda_function.lambda_handler",
    "accountId": "123456789012"
}
```

Selanjutnya

Ekstensi mengirim permintaan API `Next` untuk menerima kejadian berikutnya, yang dapat menjadi kejadian `Invoke` atau kejadian `Shutdown`. Isi respons berisi muatan, yang merupakan dokumen JSON yang berisi data kejadian.

Ekstensi mengirimkan permintaan API `Next` untuk menandakan bahwa ia siap menerima kejadian baru. Ini adalah panggilan blokir.

Jangan mengatur batas waktu pada panggilan GET karena ekstensi dapat ditangguhkan selama jangka waktu tertentu hingga ada kejadian yang akan dikembalikan.

Jalur – `/extension/event/next`

Metode – GET

Minta header

- `Lambda-Extension-Identifier` – Pengidentifikasi untuk ekstensi (string UUID). Wajib: ya. Tipe: String UID.

Header respons

- `Lambda-Extension-Event-Identifier`— Pengidentifikasi unik untuk acara (string UUID).

Kode respons

- 200 – Respons berisi informasi tentang kejadian berikutnya (`EventInvoke` atau `EventShutdown`).
- 403 – Dilarang
- 500 - Kesalahan kontainer. Non-recoverable negara. Ekstensi harus segera keluar.

Kesalahan init

Ekstensi menggunakan metode ini untuk melaporkan kesalahan inisialisasi ke Lambda. Panggil ini saat ekstensi gagal menginisialisasi setelah ia terdaftar. Setelah Lambda menerima kesalahan, tidak ada panggilan API lebih lanjut yang berhasil. Ekstensi harus keluar setelah menerima respon dari Lambda.

Jalur – `/extension/init/error`

Metode – POST

Minta header

- `Lambda-Extension-Identifier` – Pengidentifikasi unik untuk ekstensi. Wajib: ya. Tipe: String UID.
- `Lambda-Extension-Function-Error-Type` – Tipe kesalahan yang ekstensi ditemui. Wajib: ya. Header ini terdiri dari nilai string. Lambda menerima string apa pun, tetapi kami merekomendasikan format <category.reason>. Contoh:
  - Extension.NoSuchHandler
  - Extension.APIKeyNotFound
  - Extension.ConfigInvalid
  - Extension.UnknownReason

Minta parameter tubuh

- `ErrorRequest` – Informasi tentang kesalahan. Wajib: tidak.

Bidang ini adalah objek JSON dengan struktur berikut:

```
{
      errorMessage: string (text description of the error),
      errorType: string,
      stackTrace: array of strings
}
```

Perhatikan bahwa Lambda menerima nilai apa pun untuk `errorType`.

Contoh berikut menunjukkan pesan kesalahan fungsi Lambda ketika fungsi tidak dapat mengurai data peristiwa yang disediakan dalam invokasi.

Example Kesalahan fungsi

```
{
      "errorMessage" : "Error parsing event data.",
      "errorType" : "InvalidEventDataException",
      "stackTrace": [ ]
}
```

Kode respons

- 202 – Diterima
- 400 – Permintaan Buruk
- 403 – Dilarang
- 500 - Kesalahan kontainer. Non-recoverable negara. Ekstensi harus segera keluar.

Kesalahan keluar

Ekstensi menggunakan metode ini untuk melaporkan kesalahan ke Lambda sebelum keluar. Panggil ini saat Anda mengalami kegagalan tak terduga. Setelah Lambda menerima kesalahan, tidak ada panggilan API lebih lanjut yang berhasil. Ekstensi harus keluar setelah menerima respon dari Lambda.

Jalur – `/extension/exit/error`

Metode – POST

Minta header

- `Lambda-Extension-Identifier` – Pengidentifikasi unik untuk ekstensi. Wajib: ya. Tipe: String UID.
- `Lambda-Extension-Function-Error-Type` – Tipe kesalahan yang ekstensi ditemui. Wajib: ya. Header ini terdiri dari nilai string. Lambda menerima string apa pun, tetapi kami merekomendasikan format <category.reason>. Contoh:
  - Extension.NoSuchHandler
  - Extension.APIKeyNotFound
  - Extension.ConfigInvalid
  - Extension.UnknownReason

Minta parameter tubuh

- `ErrorRequest` – Informasi tentang kesalahan. Wajib: tidak.

Bidang ini adalah objek JSON dengan struktur berikut:

```
{
      errorMessage: string (text description of the error),
      errorType: string,
      stackTrace: array of strings
}
```

Perhatikan bahwa Lambda menerima nilai apa pun untuk `errorType`.

Contoh berikut menunjukkan pesan kesalahan fungsi Lambda ketika fungsi tidak dapat mengurai data peristiwa yang disediakan dalam invokasi.

Example Kesalahan fungsi

```
{
      "errorMessage" : "Error parsing event data.",
      "errorType" : "InvalidEventDataException",
      "stackTrace": [ ]
}
```

Kode respons

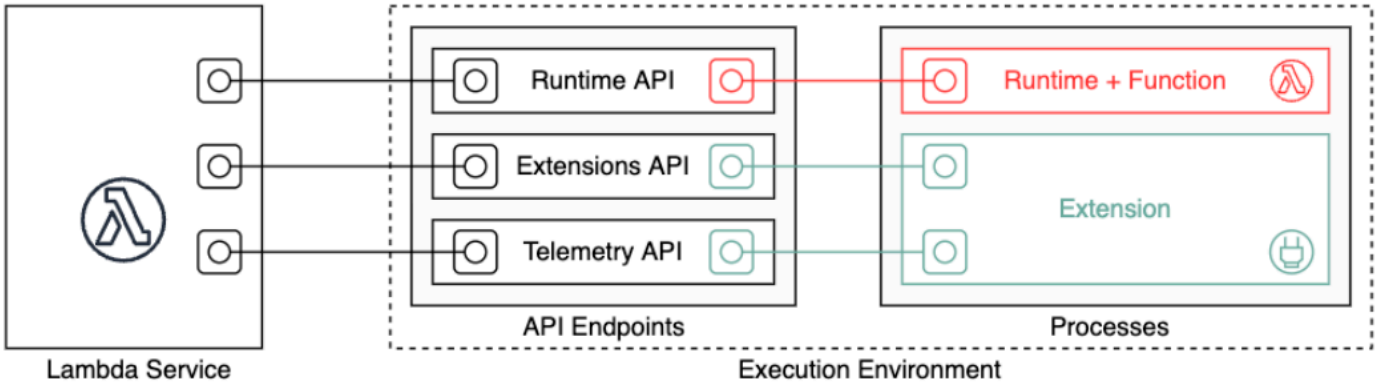
- 202 – Diterima
- 400 – Permintaan Buruk
- 403 – Dilarang
- 500 - Kesalahan kontainer. Non-recoverable negara. Ekstensi harus segera keluar.

Mengakses data telemetri real-time untuk ekstensi menggunakan API Telemetri

API Telemetri memungkinkan ekstensi Anda menerima data telemetri langsung dari Lambda. Selama inisialisasi dan pemanggilan fungsi, Lambda secara otomatis menangkap telemetri, termasuk log, metrik platform, dan jejak platform. API Telemetri memungkinkan ekstensi untuk mengakses data telemetri ini langsung dari Lambda dalam waktu dekat.

Dalam lingkungan eksekusi Lambda, Anda dapat berlangganan ekstensi Lambda Anda ke aliran telemetri. Setelah berlangganan, Lambda secara otomatis mengirimkan semua data telemetri ke ekstensi Anda. Anda kemudian memiliki fleksibilitas untuk memproses, memfilter, dan mengirimkan data ke tujuan pilihan Anda, seperti bucket Amazon Simple Storage Service (Amazon S3) atau penyedia alat observabilitas pihak ketiga.

Diagram berikut menunjukkan bagaimana Extensions API dan Telemetry API menghubungkan ekstensi ke Lambda dari dalam lingkungan eksekusi. Selain itu, Runtime API menghubungkan runtime dan Fungsi Acara Anda ke Lambda.



Important

API Telemetri Lambda menggantikan API Lambda Logs. Meskipun API Log tetap berfungsi penuh, kami sarankan hanya menggunakan API Telemetri di masa mendatang. Anda dapat berlangganan ekstensi Anda ke aliran telemetri menggunakan API Telemetri atau API Log. Setelah berlangganan menggunakan salah satu API ini, setiap upaya untuk berlangganan menggunakan API lain akan menghasilkan kesalahan.

Persyaratan versi skema Instans Terkelola Lambda

Instans Terkelola Lambda hanya mendukung versi `2025-01-29` skema API Telemetri. Saat berlangganan aliran telemetri untuk fungsi Instans Terkelola, Anda harus menggunakannya `"schemaVersion": "2025-01-29"` dalam permintaan langganan. Menggunakan versi skema sebelumnya menghasilkan peristiwa yang ditolak oleh Lambda.
Versi `2025-01-29` skema kompatibel ke belakang dan dapat digunakan dengan fungsi Lambda Managed Instances dan Lambda (default). Sebaiknya gunakan versi ini untuk semua ekstensi baru guna memastikan kompatibilitas di kedua model penerapan.

Ekstensi dapat menggunakan API Telemetri untuk berlangganan tiga aliran telemetri yang berbeda:

- Telemetri platform — Log, metrik, dan jejak, yang menjelaskan peristiwa dan kesalahan yang terkait dengan siklus hidup runtime lingkungan eksekusi, siklus hidup ekstensi, dan pemanggilan fungsi.
- Log fungsi - Log khusus yang dihasilkan oleh kode Fungsi Peristiwa Lambda.
- Log ekstensi - Log khusus yang dihasilkan oleh kode ekstensi Lambda.

Note

Lambda mengirimkan log dan metrik ke CloudWatch, dan melacak ke X-Ray (jika Anda telah mengaktifkan penelusuran), bahkan jika ekstensi berlangganan aliran telemetri.

Bagian-bagian

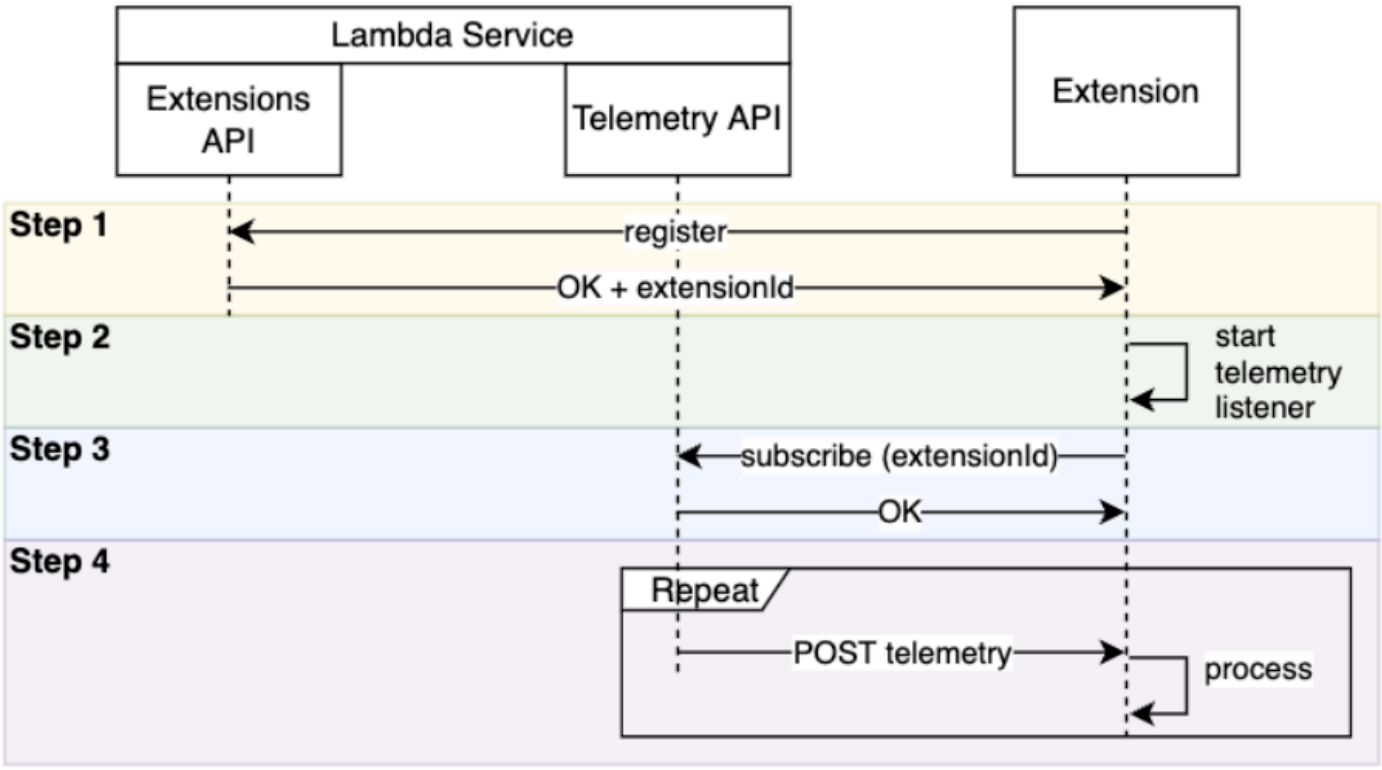
- Membuat ekstensi menggunakan API Telemetri
- Mendaftarkan ekstensi Anda
- Membuat pendengar telemetri
- Menentukan protokol tujuan
- Mengkonfigurasi penggunaan memori dan buffering
- Mengirim permintaan berlangganan ke API Telemetri
- Pesan API Telemetri Masuk
- Referensi API Telemetri Lambda
- Referensi skema Acara API Telemetri Lambda

- Mengonversi objek Lambda Telemetry API Event ke Span OpenTelemetry
- Menggunakan Lambda Logs API

Membuat ekstensi menggunakan API Telemetri

Ekstensi Lambda berjalan sebagai proses independen di lingkungan eksekusi. Ekstensi dapat terus berjalan setelah pemanggilan fungsi selesai. Karena ekstensi adalah proses yang terpisah, Anda dapat menulisnya dalam bahasa yang berbeda dari kode fungsi. Kami merekomendasikan menulis ekstensi menggunakan bahasa yang dikompilasi seperti Golang atau Rust. Dengan cara ini, ekstensi adalah biner mandiri yang dapat kompatibel dengan runtime yang didukung.

Diagram berikut menggambarkan proses empat langkah untuk membuat ekstensi yang menerima dan memproses data telemetri menggunakan API Telemetri.



Berikut adalah setiap langkah secara lebih rinci:

1. Daftarkan ekstensi Anda menggunakanthe section called “API Ekstensi”. Ini memberi Anda`Lambda-Extension-Identifier`, yang Anda perlukan dalam langkah-langkah berikut. Untuk informasi selengkapnya tentang cara mendaftarkan ekstensi Anda, lihatthe section called “Mendaftarkan ekstensi Anda”.

2. Buat pendengar telemetri. Ini bisa berupa server HTTP atau TCP dasar. Lambda menggunakan URI pendengar telemetri untuk mengirim data telemetri ke ekstensi Anda. Untuk informasi selengkapnya, lihat the section called “Membuat pendengar telemetri”.
3. Menggunakan Subscribe API di API Telemetri, berlangganan ekstensi Anda ke aliran telemetri yang diinginkan. Anda akan memerlukan URI pendengar telemetri Anda untuk langkah ini. Untuk informasi selengkapnya, lihat the section called “Mengirim permintaan berlangganan ke API Telemetri”.
4. Dapatkan data telemetri dari Lambda melalui pendengar telemetri. Anda dapat melakukan pemrosesan kustom data ini, seperti mengirimkan data ke Amazon S3 atau ke layanan observabilitas eksternal.

> **Note**
>
> Lingkungan eksekusi Event Function dapat dimulai dan berhenti beberapa kali sebagai bagian dari siklus hidupnya. Secara umum, kode ekstensi Anda berjalan selama pemanggilan Fungsi Acara, dan juga hingga 2 detik selama fase shutdown. Kami merekomendasikan pengelompokan telemetri saat sampai ke pendengar Anda. Kemudian, gunakan peristiwa `Invoke` dan `Shutdown` siklus hidup untuk mengirim setiap batch ke tujuan yang diinginkan.

Mendaftarkan ekstensi Anda

Sebelum Anda dapat berlangganan data telemetri, Anda harus mendaftarkan ekstensi Lambda Anda. Registrasi terjadi selama fase inisialisasi ekstensi. Contoh berikut menunjukkan permintaan HTTP untuk mendaftarkan ekstensi.

```
POST http://${AWS_LAMBDA_RUNTIME_API}/2020-01-01/extension/register
 Lambda-Extension-Name: lambda_extension_name
{
    'events': [ 'INVOKE', 'SHUTDOWN']
}
```

Jika permintaan berhasil, pelanggan akan menerima respons keberhasilan HTTP 200. Header respon berisi file`Lambda-Extension-Identifier`. Badan respons mengandung sifat lain dari fungsi tersebut.

```
HTTP/1.1 200 OK
Lambda-Extension-Identifier: a1b2c3d4-5678-90ab-cdef-EXAMPLE11111
```

```
{
    "functionName": "lambda_function",
    "functionVersion": "$LATEST",
    "handler": "lambda_handler",
    "accountId": "123456789012"
}
```

Untuk informasi selengkapnya, lihat the section called “Referensi API ekstensi”.

Membuat pendengar telemetri

Ekstensi Lambda Anda harus memiliki listener yang menangani permintaan masuk dari API Telemetri. Kode berikut menunjukkan contoh implementasi pendengar telemetri di Golang:

```
// Starts the server in a goroutine where the log events will be sent
func (s *TelemetryApiListener) Start() (string, error) {
 address := listenOnAddress()
 l.Info("[listener:Start] Starting on address", address)
 s.httpServer = &http.Server{Addr: address}
 http.HandleFunc("/", s.http_handler)
 go func() {
  err := s.httpServer.ListenAndServe()
  if err != http.ErrServerClosed {
   l.Error("[listener:goroutine] Unexpected stop on Http Server:", err)
   s.Shutdown()
  } else {
   l.Info("[listener:goroutine] Http Server closed:", err)
  }
 }()
 return fmt.Sprintf("http://%s/", address), nil
}

// http_handler handles the requests coming from the Telemetry API.
// Everytime Telemetry API sends log events, this function will read them from the
 response body
// and put into a synchronous queue to be dispatched later.
// Logging or printing besides the error cases below is not recommended if you have
 subscribed to
// receive extension logs. Otherwise, logging here will cause Telemetry API to send new
 logs for
// the printed lines which may create an infinite loop.
func (s *TelemetryApiListener) http_handler(w http.ResponseWriter, r *http.Request) {
 body, err := ioutil.ReadAll(r.Body)
```

```
 if err != nil {
  l.Error("[listener:http_handler] Error reading body:", err)
  return
 }

 // Parse and put the log messages into the queue
 var slice []interface{}
 _ = json.Unmarshal(body, &slice)

 for _, el := range slice {
  s.LogEventsQueue.Put(el)
 }

 l.Info("[listener:http_handler] logEvents received:", len(slice), " LogEventsQueue
 length:", s.LogEventsQueue.Len())
 slice = nil
}
```

Menentukan protokol tujuan

Saat berlangganan untuk menerima telemetri menggunakan API Telemetri, Anda dapat menentukan protokol tujuan selain URI tujuan:

```
{
    "destination": {
        "protocol": "HTTP",
        "URI": "http://sandbox.localdomain:8080"
    }
}
```

Lambda menerima dua protokol untuk menerima telemetri:

- HTTP (direkomendasikan) - Lambda mengirimkan telemetri ke titik akhir HTTP lokal (`http://sandbox.localdomain:${PORT}/${PATH}`) sebagai larik catatan dalam format JSON. Parameter `$PATH` bersifat opsional. Lambda hanya mendukung HTTP, bukan HTTPS. Lambda mengirimkan telemetri melalui permintaan POST.
- TCP — Lambda mengirimkan telemetri ke port TCP dalam format JSON (NDJSON) yang dibatasi Newline.

> **Note**
>
> Kami sangat menyarankan menggunakan HTTP daripada TCP. Dengan TCP, platform Lambda tidak dapat mengakui kapan mengirimkan telemetri ke lapisan aplikasi. Oleh karena itu, jika ekstensi Anda macet, Anda mungkin kehilangan telemetri. HTTP tidak memiliki batasan ini.

Sebelum berlangganan untuk menerima telemetri, buat pendengar HTTP lokal atau port TCP. Selama penyetelan, perhatikan hal berikut ini:

- Lambda mengirimkan telemetri hanya ke tujuan yang berada di dalam lingkungan eksekusi.
- Lambda mencoba lagi untuk mengirim telemetri (dengan backoff) tanpa adanya pendengar, atau jika permintaan POST mengalami kesalahan. Jika pendengar telemetri mogok, ia melanjutkan menerima telemetri setelah Lambda memulai ulang lingkungan eksekusi.
- Lambda mencadangkan port 9001. Tidak ada pembatasan atau rekomendasi nomor port lainnya.

Mengkonfigurasi penggunaan memori dan buffering

Penggunaan memori di lingkungan eksekusi tumbuh secara linier dengan jumlah pelanggan. Langganan mengkonsumsi sumber daya memori karena masing-masing membuka buffer memori baru untuk menyimpan data telemetri. Penggunaan memori buffer berkontribusi pada konsumsi memori secara keseluruhan di lingkungan eksekusi.

Saat berlangganan untuk menerima telemetri melalui API Telemetri, Anda memiliki opsi untuk menyangga data telemetri dan mengirimkannya ke pelanggan dalam batch. Untuk mengoptimalkan penggunaan memori, Anda dapat menentukan konfigurasi buffering:

```
{
    "buffering": {
        "maxBytes": 256*1024,
        "maxItems": 1000,
        "timeoutMs": 100
    }
}
```

Parameter	Deskripsi	Default dan batas
`maxBytes`	Volume maksimum telemetri (dalam byte) untuk buffer dalam memori.	standar: 262.144 Minimal: 262.144 Maksimal: 1.048.576
`maxItems`	Jumlah maksimum peristiwa untuk buffer dalam memori.	Default: 10.000 Minimal: 1.000 Maksimum: 10.000.
`timeoutMs`	Waktu maksimum (dalam milidetik) untuk buffer batch.	Default: 1.000 Minimal: 25 Maksimal: 30.000

Saat menyiapkan buffering, ingatlah poin-poin ini:

- Jika salah satu aliran input ditutup, Lambda akan menyiram log. Misalnya, ini dapat terjadi jika runtime mogok.
- Setiap pelanggan dapat menyesuaikan konfigurasi buffering mereka dalam permintaan berlangganan mereka.
- Saat menentukan ukuran buffer untuk membaca data, antisipasi menerima muatan sebesar2 * `maxBytes + metadataBytes`, di mana `maxBytes` merupakan komponen penyiapan buffering Anda. Untuk mengukur jumlah yang `metadataBytes` perlu dipertimbangkan, tinjau metadata berikut. Lambda menambahkan metadata yang mirip dengan ini ke setiap catatan:

```
{
   "time": "2022-08-20T12:31:32.123Z",
   "type": "function",
   "record": "Hello World"
}
```

- Jika pelanggan tidak dapat memproses telemetri masuk dengan cukup cepat, atau jika kode Fungsi Acara Anda menghasilkan volume log yang sangat tinggi, Lambda mungkin akan menjatuhkan

catatan untuk menjaga pemanfaatan memori tetap terbatas. Ketika ini terjadi, Lambda mengirimkan acara. `platform.logsDropped`

Mengirim permintaan berlangganan ke API Telemetri

Ekstensi Lambda dapat berlangganan untuk menerima data telemetri dengan mengirimkan permintaan berlangganan ke API Telemetri. Permintaan berlangganan harus berisi informasi tentang jenis acara yang Anda inginkan untuk berlangganan ekstensi. Selain itu, permintaan dapat berisi informasi tujuan pengiriman dan konfigurasi buffering.

Sebelum mengirim permintaan berlangganan, Anda harus memiliki ID ekstensi (`Lambda-Extension-Identifier`). Saat mendaftarkan ekstensi dengan Extensions API, Anda mendapatkan ID ekstensi dari respons API.

Langganan terjadi selama fase inisialisasi ekstensi. Contoh berikut menunjukkan permintaan HTTP untuk berlangganan ketiga aliran telemetri: telemetri platform, log fungsi, dan log ekstensi.

```
PUT http://${AWS_LAMBDA_RUNTIME_API}/2022-07-01/telemetry HTTP/1.1
{
   "schemaVersion": "2025-01-29",
   "types": [
        "platform",
        "function",
        "extension"
   ],
   "buffering": {
        "maxItems": 1000,
        "maxBytes": 256*1024,
        "timeoutMs": 100
   },
   "destination": {
        "protocol": "HTTP",
        "URI": "http://sandbox.localdomain:8080"
   }
}
```

Jika permintaan berhasil, maka pelanggan menerima respons sukses HTTP 200.

```
HTTP/1.1 200 OK
"OK"
```

Pesan API Telemetri Masuk

Setelah berlangganan menggunakan API Telemetri, ekstensi secara otomatis mulai menerima telemetri dari Lambda melalui permintaan POST. Setiap badan permintaan POST berisi array `Event` objek. Masing-masing `Event` memiliki skema berikut:

```
{
   time: String,
   type: String,
   record: Object
}
```

- `time`Properti menentukan kapan platform Lambda menghasilkan acara. Ini berbeda dengan saat peristiwa itu benar-benar terjadi. Nilai string `time` adalah stempel waktu dalam format ISO 8601.
- `type`Properti mendefinisikan jenis acara. Tabel berikut menjelaskan semua nilai yang mungkin.
- `record`Properti mendefinisikan objek JSON yang berisi data telemetri. Skema objek JSON ini tergantung pada. `type`

> **Pemesanan acara dengan pemanggilan bersamaan**
>
> Untuk Instans Terkelola Lambda, beberapa pemanggilan fungsi dapat dijalankan secara bersamaan dalam lingkungan eksekusi yang sama. Dalam hal ini, urutan `platform.start` dan `platform.report` acara tidak dijamin antara pemanggilan bersamaan yang berbeda. Ekstensi harus menangani peristiwa dari beberapa pemanggilan yang berjalan secara paralel dan tidak boleh mengasumsikan urutan berurutan.
> Untuk mengaitkan peristiwa dengan benar ke pemanggilan tertentu, ekstensi harus menggunakan `requestId` bidang yang ada dalam acara platform ini. Setiap pemanggilan memiliki ID permintaan unik yang tetap konsisten di semua peristiwa untuk pemanggilan itu, memungkinkan ekstensi untuk menghubungkan peristiwa dengan benar bahkan ketika mereka tiba di luar pesanan.

Tabel berikut merangkum semua jenis `Event` objek, dan menautkan ke referensi `Event` skema API Telemetri untuk setiap jenis peristiwa.

Kategori	Tipe peristiwa	Deskripsi	Skema catatan acara
Acara platform	`platform.initStart`	Inisialisasi fungsi dimulai.	the section called "`Platform.initStart`" skema
Acara platform	`platform.initRuntimeDone`	Inisialisasi fungsi selesai.	the section called "`platform.initRuntimeDone`" skema
Acara platform	`platform.initReport`	Laporan inisialisasi fungsi.	the section called "`platform.initReport`" skema
Acara platform	`platform.start`	Pemanggilan fungsi dimulai.	the section called "`platform.start`" skema
Acara platform	`platform.runtimeDone`	Runtime selesai memproses acara dengan keberhasilan atau kegagalan.	the section called "`platform.runtimeDone`" skema
Acara platform	`platform.report`	Laporan pemanggilan fungsi.	the section called "`platform.report`" skema
Acara platform	`platform.restoreStart`	Pemulihan runtime dimulai.	the section called "`platform.RestoreStart`" skema
Acara platform	`platform.restoreRuntimeDone`	Pemulihan runtime selesai.	the section called "`platform.restore`

Kategori	Tipe peristiwa	Deskripsi	Skema catatan acara
			`RuntimeDone` "skema
Acara platform	`platform.restoreReport`	Laporan pemulihan runtime.	the section called "`Platform.RestoreReport` "skema
Acara platform	`platform.telemetrySubscription`	Ekstensi berlangganan API Telemetri.	the section called "`platform.telemetrySubscription` "skema
Acara platform	`platform.logsDropped`	Lambda menjatuhkan entri log.	the section called "`platform.logsDropped` "skema
Log fungsi	`function`	Sebuah baris log dari kode fungsi.	the section called "`Fungsi`"skema
Log ekstensi:	`extension`	Sebuah baris log dari kode ekstensi.	the section called "`luas`"skema

Referensi API Telemetri Lambda

Gunakan titik akhir API Telemetri Lambda untuk berlangganan ekstensi ke aliran telemetri. Anda dapat mengambil titik akhir API Telemetri dari variabel lingkungan. `AWS_LAMBDA_RUNTIME_API` Untuk mengirim permintaan API, tambahkan versi API (`2022-07-01/`) dan`telemetry/`. Contoh:

```
http://${AWS_LAMBDA_RUNTIME_API}/2022-07-01/telemetry/
```

Untuk definisi OpenAPI Specification (OAS) dari versi respons langganan`2025-01-29`, lihat berikut ini:

- HTTP — telemetry-api-http-schema.zip
- TCP — telemetry-api-tcp-schema .zip

Operasi API

- Langganan

Langganan

Untuk berlangganan aliran telemetri, ekstensi Lambda dapat mengirim permintaan API Berlangganan.

- Jalur – `/telemetry`
- Metode - PUT
- Header
  - `Content-Type`: `application/json`
- Minta parameter tubuh
  - SchemaVersion
    - Wajib: Ya
    - Tipe: String
    - Nilai valid: `"2025-01-29"`, `"2022-12-13"`, atau `"2022-07-01"`
    - Catatan: Instans Terkelola Lambda memerlukan. `"2025-01-29"` Versi ini kompatibel dengan fungsi Lambda (default).
  - tujuan — Pengaturan konfigurasi yang menentukan tujuan acara telemetri dan protokol untuk pengiriman acara.

- Wajib: Ya
- Tipe: Objek

```
{
    "protocol": "HTTP",
    "URI": "http://sandbox.localdomain:8080"
}
```

- Protokol — Protokol yang digunakan Lambda untuk mengirim data telemetri.
  - Wajib: Ya
  - Tipe: String
  - Nilai yang valid: `"HTTP"` | `"TCP"`
- URI — URI untuk mengirim data telemetri ke.
  - Wajib: Ya
  - Tipe: String
- Untuk informasi selengkapnya, lihat the section called “Menentukan protokol tujuan”.

- jenis — Jenis telemetri yang Anda inginkan untuk berlangganan ekstensi.
  - Wajib: Ya
  - Tipe: Array string
  - Nilai yang valid: `"platform"` | `"function"` | `"extension"`
- buffering - Pengaturan konfigurasi untuk buffering acara.
  - Wajib: Tidak
  - Tipe: Objek

```
{
   "buffering": {
        "maxItems": 1000,
        "maxBytes": 256*1024,
        "timeoutMs": 100
   }
}
```

- maxItems – Jumlah maksimum peristiwa untuk buffer dalam memori.
  - Wajib: Tidak
  - Jenis: Integer

    - Default: 1.000
    - Minimal: 1.000
    - Maksimum: 10.000.
- MaxBytes — Volume maksimum telemetri (dalam byte) untuk buffer dalam memori.
    - Wajib: Tidak
    - Jenis: Integer
    - standar: 262.144
    - Minimal: 262.144
    - Maksimal: 1.048.576
- timeoutMs – Waktu maksimum (dalam milidetik) untuk mem-buffer batch.
    - Wajib: Tidak
    - Jenis: Integer
    - Default: 1.000
    - Minimal: 25
    - Maksimal: 30.000
- Untuk informasi selengkapnya, lihat the section called “Mengkonfigurasi penggunaan memori dan buffering”.

Contoh Permintaan API Berlangganan

```
PUT http://${AWS_LAMBDA_RUNTIME_API}/2022-07-01/telemetry HTTP/1.1
{
   "schemaVersion": "2025-01-29",
   "types": [
        "platform",
        "function",
        "extension"
   ],
   "buffering": {
        "maxItems": 1000,
        "maxBytes": 256*1024,
        "timeoutMs": 100
   },
   "destination": {
        "protocol": "HTTP",
        "URI": "http://sandbox.localdomain:8080"
```

```
    }
}
```

Jika permintaan Berlangganan berhasil, ekstensi menerima respons sukses HTTP 200:

```
HTTP/1.1 200 OK
"OK"
```

Jika permintaan Berlangganan gagal, ekstensi menerima respons kesalahan. Contoh:

```
HTTP/1.1 400 OK
{
    "errorType": "ValidationError",
    "errorMessage": "URI port is not provided; types should not be empty"
}
```

Berikut adalah beberapa kode respons tambahan yang dapat diterima ekstensi:

- 200 – Permintaan berhasil diselesaikan
- 202 – Permintaan diterima. Respons permintaan berlangganan di lingkungan pengujian lokal
- 400 — Permintaan buruk
- 500 – Kesalahan Layanan

Referensi skema Acara API Telemetri Lambda

Gunakan titik akhir API Telemetri Lambda untuk berlangganan ekstensi ke aliran telemetri. Anda dapat mengambil titik akhir API Telemetri dari variabel lingkungan. `AWS_LAMBDA_RUNTIME_API` Untuk mengirim permintaan API, tambahkan versi API (`2022-07-01/`) dan`telemetry/`. Contoh:

```
http://${AWS_LAMBDA_RUNTIME_API}/2022-07-01/telemetry/
```

Untuk definisi OpenAPI Specification (OAS) dari versi respons langganan`2025-01-29`, lihat berikut ini:

- HTTP - telemetry-api-http-schema.zip
- TCP - telemetry-api-tcp-schema.zip

Tabel berikut adalah ringkasan dari semua jenis `Event` objek yang didukung oleh API Telemetri.

Kategori	Tipe peristiwa	Deskripsi	Skema catatan acara
Acara platform	`platform.initStart`	Inisialisasi fungsi dimulai.	the section called “`Platform.initStart`”skema
Acara platform	`platform.initRuntimeDone`	Inisialisasi fungsi selesai.	the section called “`platform.initRuntimeDone`”skema
Acara platform	`platform.initReport`	Laporan inisialisasi fungsi.	the section called “`platform.initReport`”skema
Acara platform	`platform.start`	Pemanggilan fungsi dimulai.	the section called “`platform.start`”skema
Acara platform	`platform.runtimeDone`	Runtime selesai memproses acara	the section called “`platform.`

Kategori	Tipe peristiwa	Deskripsi	Skema catatan acara
		dengan keberhasilan atau kegagalan.	`runtimeDone` "skema
Acara platform	`platform.report`	Laporan pemanggilan fungsi.	the section called "`platform.report`" skema
Acara platform	`platform.restoreStart`	Pemulihan runtime dimulai.	the section called "`platform.RestoreStart`" skema
Acara platform	`platform.restoreRuntimeDone`	Pemulihan runtime selesai.	the section called "`platform.restoreRuntimeDone`" skema
Acara platform	`platform.restoreReport`	Laporan pemulihan runtime.	the section called "`Platform.RestoreReport`" skema
Acara platform	`platform.telemetrySubscription`	Ekstensi berlangganan API Telemetri.	the section called "`platform.telemetrySubscription`" skema
Acara platform	`platform.logsDropped`	Lambda menjatuhkan entri log.	the section called "`platform.logsDropped`" skema
Log fungsi	`function`	Sebuah baris log dari kode fungsi.	the section called "`Fungsi`" skema

Kategori	Tipe peristiwa	Deskripsi	Skema catatan acara
Log ekstensi:	`extension`	Sebuah baris log dari kode ekstensi.	the section called "`luas`"skema

Daftar Isi

- Jenis objek Acara API Telemetri
  - Platform.initStart
  - platform.init RuntimeDone
  - platform.initReport
  - platform.start
  - platform.runtimeDone
  - platform.report
  - platform.RestoreStart
  - platform.restore RuntimeDone
  - Platform.RestoreReport
  - platform.extension
  - platform.telemetrySubscription
  - platform.logsDropped
  - Fungsi
  - luas
- Jenis objek bersama
  - InitPhase
  - InitReportMetrics
  - InitType
  - ReportMetrics
  - RestoreReportMetrics
  - RuntimeDoneMetrics
  - Rentang
  - Status
  - TraceContext

- TracingType

Jenis objek **`Acara`** API Telemetri

Bagian ini merinci jenis `Event` objek yang didukung oleh API Telemetri Lambda. Dalam deskripsi peristiwa, tanda tanya (?) menunjukkan bahwa atribut mungkin tidak ada dalam objek.

`Platform.initStart`

Sebuah `platform.initStart` peristiwa menunjukkan bahwa fase inisialisasi Fungsi Peristiwa telah dimulai. Sebuah `platform.initStart` `Event` objek memiliki bentuk sebagai berikut:

```
Event: Object
- time: String
- type: String = platform.initStart
- record: PlatformInitStart
```

`PlatformInitStart`Objek memiliki atribut berikut:

- FunctionName - `String`
- FunctionVersion - `String`
- InitializationType - objek `the section called “InitType”`
- InstanceId? — `String`
- contohMaxMemory? — `Integer`
- fase — `the section called “InitPhase”` objek
- RuntimeVersion? — `String`
- runtime? VersionArn — `String`

Berikut ini adalah contoh `Event` tipe`platform.initStart`:

```
{
    "time": "2022-10-12T00:00:15.064Z",
    "type": "platform.initStart",
    "record": {
        "initializationType": "on-demand",
        "phase": "init",
```

```
        "runtimeVersion": "nodejs-14.v3",
        "runtimeVersionArn": "arn",
        "functionName": "myFunction",
        "functionVersion": "$LATEST",
        "instanceId": "82561ce0-53dd-47d1-90e0-c8f5e063e62e",
        "instanceMaxMemory": 256
    }
}
```

`platform.init RuntimeDone`

Sebuah `platform.initRuntimeDone` peristiwa menunjukkan bahwa fase inisialisasi Fungsi Peristiwa telah selesai. Sebuah `platform.initRuntimeDone` `Event` objek memiliki bentuk sebagai berikut:

```
Event: Object
- time: String
- type: String = platform.initRuntimeDone
- record: PlatformInitRuntimeDone
```

`PlatformInitRuntimeDone`Objek memiliki atribut berikut:

- InitializationType - objek the section called “InitType”
- fase — the section called “InitPhase” objek
- status - the section called “Status” objek
- bentang? — Daftar the section called “Rentang” objek

Berikut ini adalah contoh `Event` tipe`platform.initRuntimeDone`:

```
{
    "time": "2022-10-12T00:01:15.000Z",
    "type": "platform.initRuntimeDone",
    "record": {
        "initializationType": "on-demand"
        "status": "success",
        "spans": [
            {
                "name": "someTimeSpan",
                "start": "2022-06-02T12:02:33.913Z",
                "durationMs": 70.5
```

```
            }
        ]
    }
}
```

`platform.initReport`

Sebuah `platform.initReport` peristiwa berisi laporan keseluruhan dari fase inisialisasi Fungsi Peristiwa. Sebuah `platform.initReport` `Event` objek memiliki bentuk sebagai berikut:

```
Event: Object
- time: String
- type: String = platform.initReport
- record: PlatformInitReport
```

`PlatformInitReport`Objek memiliki atribut berikut:

- ErrorType? — tali
- InitializationType - objek the section called “InitType”
- fase — the section called “InitPhase” objek
- metrik — objek the section called “InitReportMetrics”
- bentang? — Daftar the section called “Rentang” objek
- status - the section called “Status” objek

Berikut ini adalah contoh `Event` tipe`platform.initReport`:

```
{
    "time": "2022-10-12T00:01:15.000Z",
    "type": "platform.initReport",
    "record": {
        "initializationType": "on-demand",
        "status": "success",
        "phase": "init",
        "metrics": {
            "durationMs": 125.33
        },
        "spans": [
            {
```

```
                "name": "someTimeSpan",
                "start": "2022-06-02T12:02:33.913Z",
                "durationMs": 90.1
            }
        ]
    }
}
```

`platform.start`

Sebuah `platform.start` peristiwa menunjukkan bahwa fase pemanggilan Fungsi Acara telah dimulai. Sebuah `platform.start` `Event` objek memiliki bentuk sebagai berikut:

```
Event: Object
- time: String
- type: String = platform.start
- record: PlatformStart
```

`PlatformStart`Objek memiliki atribut berikut:

- RequesTid — `String`
- versi? — `String`
- melacak? — [the section called “TraceContext”]

Berikut ini adalah contoh `Event` tipe`platform.start`:

```
{
    "time": "2022-10-12T00:00:15.064Z",
    "type": "platform.start",
    "record": {
        "requestId": "6d68ca91-49c9-448d-89b8-7ca3e6dc66aa",
        "version": "$LATEST",
        "tracing": {
            "spanId": "54565fb41ac79632",
            "type": "X-Amzn-Trace-Id",
            "value":
 "Root=1-62e900b2-710d76f009d6e7785905449a;Parent=0efbd19962d95b05;Sampled=1"
        }
    }
}
```

`platform.runtimeDone`

Sebuah `platform.runtimeDone` peristiwa menunjukkan bahwa fase pemanggilan Fungsi Peristiwa telah selesai. Sebuah `platform.runtimeDone` `Event` objek memiliki bentuk sebagai berikut:

> **Instans Terkelola Lambda**
>
> `platform.runtimeDone`Acara ini tidak didukung untuk Instans Terkelola Lambda. Ekstensi yang berjalan pada Instans Terkelola tidak menerima acara ini karena ekstensi tidak dapat berlangganan `INVOKE` acara di Instans Terkelola. Karena model eksekusi bersamaan di mana beberapa pemanggilan dapat diproses secara bersamaan, ekstensi tidak dapat melakukan pemrosesan pasca-pemanggilan untuk pemanggilan individu seperti yang biasanya dilakukan pada fungsi Lambda (default).
> Untuk `responseDuration` Instans Terkelola, rentang `responseLatency` dan yang biasanya disertakan dalam `platform.runtimeDone` acara tersebut `platform.report` tersedia. Lihat the section called “`platform.report`” untuk detail.

```
Event: Object
- time: String
- type: String = platform.runtimeDone
- record: PlatformRuntimeDone
```

`PlatformRuntimeDone`Objek memiliki atribut berikut:

- ErrorType? — `String`
- metrik? — `the section called “RuntimeDoneMetrics”` objek
- RequesTid — `String`
- status - `the section called “Status”` objek
- bentang? — Daftar `the section called “Rentang”` objek
- melacak? — `the section called “TraceContext”` objek

Berikut ini adalah contoh `Event` tipe`platform.runtimeDone`:

```
{
```

```
    "time": "2022-10-12T00:01:15.000Z",
    "type": "platform.runtimeDone",
    "record": {
        "requestId": "6d68ca91-49c9-448d-89b8-7ca3e6dc66aa",
        "status": "success",
        "tracing": {
            "spanId": "54565fb41ac79632",
            "type": "X-Amzn-Trace-Id",
            "value":
 "Root=1-62e900b2-710d76f009d6e7785905449a;Parent=0efbd19962d95b05;Sampled=1"
        },
        "spans": [
            {
                "name": "someTimeSpan",
                "start": "2022-08-02T12:01:23:521Z",
                "durationMs": 80.0
            }
        ],
        "metrics": {
            "durationMs": 140.0,
            "producedBytes": 16
        }
    }
}
```

`platform.report`

Sebuah `platform.report` peristiwa berisi laporan keseluruhan dari fase pemanggilan Fungsi Peristiwa. Sebuah `platform.report` `Event` objek memiliki bentuk sebagai berikut:

> **Instans Terkelola Lambda**
>
> `platform.report`Acara untuk Instans Terkelola Lambda memiliki metrik dan rentang yang berbeda dibandingkan dengan fungsi Lambda (default). Untuk Instans Terkelola:
>
> - Rentang: Berisi `responseLatency` dan `responseDuration` bukannya. `extensionOverhead` `extensionOverhead`Rentang tidak tersedia karena ekstensi tidak dapat berlangganan `INVOKE` acara pada Instans Terkelola karena model eksekusi bersamaan.
> - Metrik: Hanya termasuk`durationMs`. Metrik berikut tidak termasuk:`billedDurationMs`,, `initDurationMs``maxMemoryUsedMB`, dan`memorySizeMB`. Metrik per-pemanggilan ini

tidak berlaku di lingkungan eksekusi bersamaan. Untuk metrik pemanfaatan sumber daya, gunakan atau Wawasan Memantau Instans Terkelola Lambda Lambda.

```
Event: Object
- time: String
- type: String = platform.report
- record: PlatformReport
```

`PlatformReport`Objek memiliki atribut berikut:

- metrik — objek `the section called “ReportMetrics”`
- RequesTid — `String`
- bentang? — Daftar `the section called “Rentang”` objek
- status - `the section called “Status”` objek
- menelusuri? — `the section called “TraceContext”` objek

Berikut ini adalah contoh `Event` tipe`platform.report`:

```
{
    "time": "2022-10-12T00:01:15.000Z",
    "type": "platform.report",
    "record": {
        "metrics": {
            "billedDurationMs": 694,
            "durationMs": 693.92,
            "initDurationMs": 397.68,
            "maxMemoryUsedMB": 84,
            "memorySizeMB": 128
        },
        "requestId": "6d68ca91-49c9-448d-89b8-7ca3e6dc66aa",
    }
}
```

`platform.RestoreStart`

Sebuah `platform.restoreStart` peristiwa menunjukkan bahwa acara restorasi lingkungan Fungsi Acara dimulai. Dalam acara restorasi lingkungan, Lambda menciptakan lingkungan dari

snapshot yang di-cache daripada menginisialisasi dari awal. Untuk informasi selengkapnya, lihat SnapStart. Sebuah `platform.restoreStart` `Event` objek memiliki bentuk sebagai berikut:

```
Event: Object
- time: String
- type: String = platform.restoreStart
- record: PlatformRestoreStart
```

`PlatformRestoreStart`Objek memiliki atribut berikut:

- FunctionName - `String`
- FunctionVersion - `String`
- InstanceId? — `String`
- contohMaxMemory? — `String`
- RuntimeVersion? — `String`
- runtime? VersionArn — `String`

Berikut ini adalah contoh `Event` tipe`platform.restoreStart`:

```
{
    "time": "2022-10-12T00:00:15.064Z",
    "type": "platform.restoreStart",
    "record": {
        "runtimeVersion": "nodejs-14.v3",
        "runtimeVersionArn": "arn",
        "functionName": "myFunction",
        "functionVersion": "$LATEST",
        "instanceId": "82561ce0-53dd-47d1-90e0-c8f5e063e62e",
        "instanceMaxMemory": 256
    }
}
```

`platform.restore RuntimeDone`

Sebuah `platform.restoreRuntimeDone` peristiwa menunjukkan bahwa acara restorasi lingkungan Fungsi Acara selesai. Dalam acara restorasi lingkungan, Lambda menciptakan lingkungan dari snapshot yang di-cache daripada menginisialisasi dari awal. Untuk informasi selengkapnya, lihat SnapStart. Sebuah `platform.restoreRuntimeDone` `Event` objek memiliki bentuk sebagai berikut:

```
Event: Object
- time: String
- type: String = platform.restoreRuntimeDone
- record: PlatformRestoreRuntimeDone
```

`PlatformRestoreRuntimeDone`Objek memiliki atribut berikut:

- ErrorType? — `String`
- bentang? — Daftar the section called "Rentang" objek
- status - the section called "Status" objek

Berikut ini adalah contoh `Event` tipe`platform.restoreRuntimeDone`:

```
{
    "time": "2022-10-12T00:00:15.064Z",
    "type": "platform.restoreRuntimeDone",
    "record": {
        "status": "success",
        "spans": [
            {
                "name": "someTimeSpan",
                "start": "2022-08-02T12:01:23:521Z",
                "durationMs": 80.0
            }
        ]
    }
}
```

`Platform.RestoreReport`

Sebuah `platform.restoreReport` acara berisi laporan keseluruhan dari acara restorasi Fungsi Acara. Sebuah `platform.restoreReport` `Event` objek memiliki bentuk sebagai berikut:

```
Event: Object
- time: String
- type: String = platform.restoreReport
- record: PlatformRestoreReport
```

`PlatformRestoreReport`Objek memiliki atribut berikut:

- ErrorType? — tali
- metrik? — the section called “RestoreReportMetrics” objek
- bentang? — Daftar the section called “Rentang” objek
- status - the section called “Status” objek

Berikut ini adalah contoh Event tipeplatform.restoreReport:

```
{
    "time": "2022-10-12T00:00:15.064Z",
    "type": "platform.restoreReport",
    "record": {
        "status": "success",
        "metrics": {
            "durationMs": 15.19
        },
        "spans": [
            {
                "name": "someTimeSpan",
                "start": "2022-08-02T12:01:23:521Z",
                "durationMs": 30.0
            }
        ]
    }
}
```

platform.extension

Sebuah extension peristiwa berisi log dari kode ekstensi. Sebuah extension Event objek memiliki bentuk sebagai berikut:

```
Event: Object
- time: String
- type: String = extension
- record: {}
```

PlatformExtensionObjek memiliki atribut berikut:

- event — Daftar String
- nama — String
- negara — String

Berikut ini adalah contoh Event tipeplatform.extension:

```
{
    "time": "2022-10-12T00:02:15.000Z",
    "type": "platform.extension",
    "record": {
        "events": [ "INVOKE", "SHUTDOWN" ],
        "name": "my-telemetry-extension",
        "state": "Ready"
    }
}
```

platform.telemetrySubscription

Sebuah platform.telemetrySubscription acara berisi informasi tentang langganan ekstensi. Sebuah platform.telemetrySubscription Event objek memiliki bentuk sebagai berikut:

```
Event: Object
- time: String
- type: String = platform.telemetrySubscription
- record: PlatformTelemetrySubscription
```

PlatformTelemetrySubscriptionObjek memiliki atribut berikut:

- nama — String
- negara — String
- jenis — Daftar String

Berikut ini adalah contoh Event tipeplatform.telemetrySubscription:

```
{
    "time": "2022-10-12T00:02:35.000Z",
    "type": "platform.telemetrySubscription",
    "record": {
        "name": "my-telemetry-extension",
        "state": "Subscribed",
        "types": [ "platform", "function" ]
    }
}
```

platform.logsDropped

Sebuah `platform.logsDropped` acara berisi informasi tentang peristiwa yang dijatuhkan. Lambda memancarkan `platform.logsDropped` peristiwa ketika Fungsi Peristiwa mengeluarkan log pada tingkat yang terlalu tinggi bagi Lambda untuk memprosesnya. Ketika Lambda tidak dapat mengirim log ke CloudWatch atau ke ekstensi yang berlangganan Telemetri API dengan kecepatan yang dihasilkan oleh Event Function, log akan turun untuk mencegah eksekusi Event Function melambat. Sebuah `platform.logsDropped` `Event` objek memiliki bentuk sebagai berikut:

```
Event: Object
- time: String
- type: String = platform.logsDropped
- record: PlatformLogsDropped
```

`PlatformLogsDropped`Objek memiliki atribut berikut:

- DroppedBytes — `Integer`
- DroppedRecords — `Integer`
- alasan — `String`

Berikut ini adalah contoh `Event` tipe`platform.logsDropped`:

```
{
    "time": "2022-10-12T00:02:35.000Z",
    "type": "platform.logsDropped",
    "record": {
        "droppedBytes": 12345,
        "droppedRecords": 123,
        "reason": "Some logs were dropped because the downstream consumer is slower
 than the logs production rate"
    }
}
```

Fungsi

Sebuah `function` peristiwa berisi log dari kode Event Function. Sebuah `function` `Event` objek memiliki bentuk sebagai berikut:

```
Event: Object
- time: String
```

```
- type: String = function
- record: {}
```

Format `record` bidang tergantung pada apakah log Fungsi Acara Anda diformat dalam teks biasa atau format JSON. untuk mempelajari lebih lanjut tentang opsi konfigurasi format log, lihat the section called “Format log”

Berikut ini adalah contoh `Event` jenis `function` di mana format log adalah teks biasa:

```
{
    "time": "2022-10-12T00:03:50.000Z",
    "type": "function",
    "record": "[INFO] Hello world, I am a function!"
}
```

Berikut ini adalah contoh `Event` jenis `function` di mana format log adalah JSON:

```
{
    "time": "2022-10-12T00:03:50.000Z",
    "type": "function",
    "record": {
        "timestamp": "2022-10-12T00:03:50.000Z",
        "level": "INFO",
        "requestId": "79b4f56e-95b1-4643-9700-2807f4e68189",
        "message": "Hello world, I am a function!"
    }
}
```

> **Note**
>
> Jika versi skema yang Anda gunakan lebih lama dari `2022-12-13` versi, maka selalu `"record"` dirender sebagai string bahkan ketika format logging Fungsi Acara Anda dikonfigurasi sebagai JSON. Untuk Instans Terkelola Lambda, Anda harus menggunakan versi skema. `2025-01-29`

`luas`

Sebuah `extension` peristiwa berisi log dari kode ekstensi. Sebuah `extension` `Event` objek memiliki bentuk sebagai berikut:

```
Event: Object
- time: String
- type: String = extension
- record: {}
```

Format `record` bidang tergantung pada apakah log Fungsi Acara Anda diformat dalam teks biasa atau format JSON. untuk mempelajari lebih lanjut tentang opsi konfigurasi format log, lihat the section called “Format log”

Berikut ini adalah contoh `Event` jenis `extension` di mana format log adalah teks biasa:

```
{
    "time": "2022-10-12T00:03:50.000Z",
    "type": "extension",
    "record": "[INFO] Hello world, I am an extension!"
}
```

Berikut ini adalah contoh `Event` jenis `extension` di mana format log adalah JSON:

```
{
    "time": "2022-10-12T00:03:50.000Z",
    "type": "extension",
    "record": {
       "timestamp": "2022-10-12T00:03:50.000Z",
       "level": "INFO",
       "requestId": "79b4f56e-95b1-4643-9700-2807f4e68189",
       "message": "Hello world, I am an extension!"
    }
}
```

> **Note**
>
> Jika versi skema yang Anda gunakan lebih lama dari `2022-12-13` versi, maka selalu `"record"` dirender sebagai string bahkan ketika format logging Fungsi Acara Anda dikonfigurasi sebagai JSON. Untuk Instans Terkelola Lambda, Anda harus menggunakan versi skema. `2025-01-29`

Jenis objek bersama

Bagian ini merinci jenis objek bersama yang didukung oleh API Telemetri Lambda.

`InitPhase`

Sebuah string enum yang menggambarkan fase ketika langkah inisialisasi terjadi. Dalam kebanyakan kasus, Lambda menjalankan kode inisialisasi Fungsi Acara selama fase. `init` Namun, dalam beberapa kasus kesalahan, Lambda dapat menjalankan kembali kode inisialisasi Fungsi Peristiwa selama fase. `invoke` (Ini disebut init yang ditekan.)

- Jenis - `String`
- Nilai yang valid - `init` | `invoke` | `snap-start`

`InitReportMetrics`

Objek yang berisi metrik tentang fase inisialisasi.

- Jenis - `Object`

Sebuah `InitReportMetrics` objek memiliki bentuk sebagai berikut:

```
InitReportMetrics: Object
- durationMs: Double
```

Berikut ini adalah contoh `InitReportMetrics` objek:

```
{
    "durationMs": 247.88
}
```

`InitType`

Sebuah string enum yang menjelaskan bagaimana Lambda menginisialisasi lingkungan.

- Jenis - `String`
- Nilai yang valid - `on-demand` | `provisioned-concurrency`

`ReportMetrics`

Objek yang berisi metrik tentang fase selesai.

- Jenis - `Object`

Sebuah `ReportMetrics` objek memiliki bentuk sebagai berikut:

```
ReportMetrics: Object
- billedDurationMs: Integer
- durationMs: Double
- initDurationMs?: Double
- maxMemoryUsedMB: Integer
- memorySizeMB: Integer
- restoreDurationMs?: Double
```

Berikut ini adalah contoh `ReportMetrics` objek:

```
{
    "billedDurationMs": 694,
    "durationMs": 693.92,
    "initDurationMs": 397.68,
    "maxMemoryUsedMB": 84,
    "memorySizeMB": 128
}
```

`RestoreReportMetrics`

Objek yang berisi metrik tentang fase restorasi selesai.

- Jenis - `Object`

Sebuah `RestoreReportMetrics` objek memiliki bentuk sebagai berikut:

```
RestoreReportMetrics: Object
- durationMs: Double
```

Berikut ini adalah contoh `RestoreReportMetrics` objek:

```
{
```

```
    "durationMs": 15.19
}
```

`RuntimeDoneMetrics`

Objek yang berisi metrik tentang fase pemanggilan.

- Jenis - `Object`

Sebuah `RuntimeDoneMetrics` objek memiliki bentuk sebagai berikut:

```
RuntimeDoneMetrics: Object
- durationMs: Double
- producedBytes?: Integer
```

Berikut ini adalah contoh `RuntimeDoneMetrics` objek:

```
{
    "durationMs": 200.0,
    "producedBytes": 15
}
```

`Rentang`

Objek yang berisi detail tentang rentang. Rentang mewakili unit kerja atau operasi dalam jejak. Untuk informasi selengkapnya tentang rentang, lihat Span di halaman Tracing API di situs web OpenTelemetry Dokumen.

Lambda mendukung rentang berikut untuk acara tersebut: `platform.RuntimeDone`

- `responseLatency`Rentang ini menjelaskan berapa lama waktu yang dibutuhkan Fungsi Peristiwa Lambda Anda untuk mulai mengirim respons.
- `responseDuration`Rentang ini menjelaskan berapa lama waktu yang dibutuhkan Fungsi Peristiwa Lambda Anda untuk menyelesaikan pengiriman seluruh respons.
- `runtimeOverhead`Rentang menjelaskan berapa lama waktu yang dibutuhkan runtime Lambda untuk memberi sinyal bahwa ia siap untuk memproses pemanggilan Fungsi Peristiwa berikutnya. Ini adalah waktu yang dibutuhkan runtime untuk memanggil API pemanggilan berikutnya untuk mendapatkan acara berikutnya setelah mengembalikan respons Fungsi Acara Anda.

Berikut ini adalah contoh objek `responseLatency` span:

```
{
        "name": "responseLatency",
        "start": "2022-08-02T12:01:23.521Z",
        "durationMs": 23.02
      }
```

`Status`

Objek yang menggambarkan status fase inisialisasi atau pemanggilan. Jika statusnya salah satu `failure` atau`error`, maka `Status` objek juga berisi `errorType` bidang yang menjelaskan kesalahan.

- Jenis - `Object`
- Nilai status yang valid - `success` | `failure` | `error` | `timeout`

`TraceContext`

Objek yang menggambarkan sifat-sifat jejak.

- Jenis - `Object`

Sebuah `TraceContext` objek memiliki bentuk sebagai berikut:

```
TraceContext: Object
- spanId?: String
- type: TracingType enum
- value: String
```

Berikut ini adalah contoh `TraceContext` objek:

```
{
    "spanId": "073a49012f3c312e",
    "type": "X-Amzn-Trace-Id",
    "value":
 "Root=1-62e900b2-710d76f009d6e7785905449a;Parent=0efbd19962d95b05;Sampled=1"
}
```

`TracingType`

Sebuah string enum yang menggambarkan jenis tracing dalam suatu the section called "TraceContext" objek.

- Jenis - `String`
- Nilai yang valid - `X-Amzn-Trace-Id`

Mengonversi objek Lambda Telemetry API Event ke Span OpenTelemetry

Skema API AWS Lambda Telemetri secara semantik kompatibel dengan (OTel). OpenTelemetry Ini berarti Anda dapat mengonversi `Event` objek API AWS Lambda Telemetri Anda ke OpenTelemetry (OTel) Span. Saat mengonversi, Anda tidak boleh memetakan satu `Event` objek ke satu OTel Span.

Sebagai gantinya, Anda harus menyajikan ketiga peristiwa yang terkait dengan fase siklus hidup dalam satu Rentang OTel. Misalnya, `startruntimeDone`, dan `runtimeReport` peristiwa mewakili satu pemanggilan Fungsi Acara. Hadirkan ketiga acara ini sebagai satu OTel Span.

Anda dapat mengonversi acara menggunakan Span Events atau Child (nested) Span. Tabel di halaman ini menjelaskan pemetaan antara properti skema API Telemetri dan properti OTel Span untuk kedua pendekatan. Untuk informasi selengkapnya tentang OTel Spans, lihat Span di halaman Tracing API di situs web Dokumen. OpenTelemetry

Bagian-bagian

- Peta ke Rentang OTel dengan Acara Span
- Peta ke Otel Rentang dengan Rentang Anak

Peta ke Rentang OTel dengan Acara Span

Dalam tabel berikut, e mewakili peristiwa yang berasal dari sumber telemetri.

Memetakan acara *Mulai

OpenTelemetry	Skema API Telemetri Lambda
`Span.Name`	Ekstensi Anda menghasilkan nilai ini berdasarkan `type` bidang.
`Span.StartTime`	Gunakan `e.time`.
`Span.EndTime`	N/A, karena acara belum selesai.
`Span.Kind`	Setel ke`Server`.
`Span.Status`	Setel ke`Unset`.

OpenTelemetry	Skema API Telemetri Lambda
`Span.TraceId`	Parse AWS X-Ray header yang ditemukan di`e.tracing.value`, lalu gunakan `TraceId` nilainya.
`Span.ParentId`	Parse X-Ray header yang ditemukan di`e.tracing.value`, lalu gunakan `Parent` nilainya.
`Span.SpanId`	Gunakan `e.tracing.spanId` jika tersedia. Jika tidak, buat yang baru`SpanId`.
`Span.SpanContext.TraceState`	N/A untuk konteks X-Ray jejak.
`Span.SpanContext.TraceFlags`	Parse X-Ray header yang ditemukan di`e.tracing.value`, lalu gunakan `Sampled` nilainya.
`Span.Attributes`	Ekstensi Anda dapat menambahkan nilai kustom apa pun di sini.

Memetakan acara* RuntimeDone

OpenTelemetry	Skema API Telemetri Lambda
`Span.Name`	Ekstensi Anda menghasilkan nilai berdasarkan `type` bidang.
`Span.StartTime`	Gunakan `e.time` dari `*Start` acara yang cocok. Alternatif lainnya, gunakan `e.time - e.metrics.durationMs`.
`Span.EndTime`	N/A, karena acara belum selesai.
`Span.Kind`	Setel ke`Server`.

OpenTelemetry	Skema API Telemetri Lambda
`Span.Status`	Jika `e.status` tidak sama`success`, maka atur ke`Error`. Jika tidak, atur ke`0k`.
`Span.Events[]`	Gunakan `e.spans[]` .
`Span.Events[i].Name`	Gunakan `e.spans[i].name` .
`Span.Events[i].Time`	Gunakan `e.spans[i].start` .
`Span.TraceId`	Parse AWS X-Ray header yang ditemukan die`.tracing.value` , lalu gunakan `TraceId` nilainya.
`Span.ParentId`	Parse X-Ray header yang ditemukan die`.tracing.value` , lalu gunakan `Parent` nilainya.
`Span.SpanId`	Gunakan hal yang sama `SpanId` dari `*Start` acara tersebut. Jika tidak tersedia, maka gunakane`.tracing.spanId` , atau buat yang baru`SpanId`.
`Span.SpanContext.TraceState`	N/A untuk konteks X-Ray jejak.
`Span.SpanContext.TraceFlags`	Parse X-Ray header yang ditemukan die`.tracing.value` , lalu gunakan `Sampled` nilainya.
`Span.Attributes`	Ekstensi Anda dapat menambahkan nilai kustom apa pun di sini.

Memetakan *Laporkan peristiwa

OpenTelemetry	Skema API Telemetri Lambda
`Span.Name`	Ekstensi Anda menghasilkan nilai berdasarkan `type` bidang.
`Span.StartTime`	Gunakan `e.time` dari `*Start` acara yang cocok. Alternatif lainnya, gunakan `e.time - e.metrics.durationMs` .
`Span.EndTime`	Gunakan `e.time`.
`Span.Kind`	Setel ke`Server`.
`Span.Status`	Gunakan nilai yang sama dengan `*RuntimeDone` acara.
`Span.TraceId`	Parse AWS X-Ray header yang ditemukan die`.tracing.value` , lalu gunakan `TraceId` nilainya.
`Span.ParentId`	Parse X-Ray header yang ditemukan die`.tracing.value` , lalu gunakan `Parent` nilainya.
`Span.SpanId`	Gunakan hal yang sama `SpanId` dari `*Start` acara tersebut. Jika tidak tersedia, maka gunakane`.tracing.spanId` , atau buat yang baru`SpanId`.
`Span.SpanContext.TraceState`	N/A untuk konteks X-Ray jejak.
`Span.SpanContext.TraceFlags`	Parse X-Ray header yang ditemukan die`.tracing.value` , lalu gunakan `Sampled` nilainya.
`Span.Attributes`	Ekstensi Anda dapat menambahkan nilai kustom apa pun di sini.

Peta ke Otel Rentang dengan Rentang Anak

Tabel berikut menjelaskan cara mengonversi peristiwa API Telemetri Lambda menjadi Tel Spans with Child (nested) Span for Span. `*RuntimeDone` Untuk `*Start` dan `*Report` pemetaan, lihat tabel dithe section called “Peta ke Rentang OTel dengan Acara Span”, karena sama untuk Child Spans. Dalam tabel ini, e mewakili peristiwa yang berasal dari sumber telemetri.

Memetakan acara* RuntimeDone

OpenTelemetry	Skema API Telemetri Lambda
`Span.Name`	Ekstensi Anda menghasilkan nilai berdasarkan `type` bidang.
`Span.StartTime`	Gunakan `e.time` dari `*Start` acara yang cocok. Alternatif lainnya, gunakan `e.time - e.metrics.durationMs` .
`Span.EndTime`	N/A, karena acara belum selesai.
`Span.Kind`	Setel ke`Server`.
`Span.Status`	Jika `e.status` tidak sama`success`, maka atur ke`Error`. Jika tidak, atur ke`Ok`.
`Span.TraceId`	Parse AWS X-Ray header yang ditemukan die`.tracing.value` , lalu gunakan `TraceId` nilainya.
`Span.ParentId`	Parse X-Ray header yang ditemukan die`.tracing.value` , lalu gunakan `Parent` nilainya.
`Span.SpanId`	Gunakan hal yang sama `SpanId` dari `*Start` acara tersebut. Jika tidak tersedia, maka

OpenTelemetry	Skema API Telemetri Lambda
	gunakane.tracing.spanId , atau buat yang baruSpanId.
Span.SpanContext.TraceState	N/A untuk konteks X-Ray jejak.
Span.SpanContext.TraceFlags	Parse X-Ray header yang ditemukan die.tracing.value , lalu gunakan Sampled nilainya.
Span.Attributes	Ekstensi Anda dapat menambahkan nilai kustom apa pun di sini.
ChildSpan[i].Name	Gunakan e.spans[i].name .
ChildSpan[i].StartTime	Gunakan e.spans[i].start .
ChildSpan[i].EndTime	Gunakan e.spans[i].start + e.spans[i].durations .
ChildSpan[i].Kind	Sama seperti orang tuaSpan.Kind .
ChildSpan[i].Status	Sama seperti orang tuaSpan.Status .
ChildSpan[i].TraceId	Sama seperti orang tuaSpan.TraceId .
ChildSpan[i].ParentId	Gunakan orang tuaSpan.SpanId .
ChildSpan[i].SpanId	Hasilkan yang baruSpanId.
ChildSpan[i].SpanContext.TraceState	N/A untuk konteks X-Ray jejak.
ChildSpan[i].SpanContext.TraceFlags	Sama seperti orang tuaSpan.SpanContext.TraceFlags .

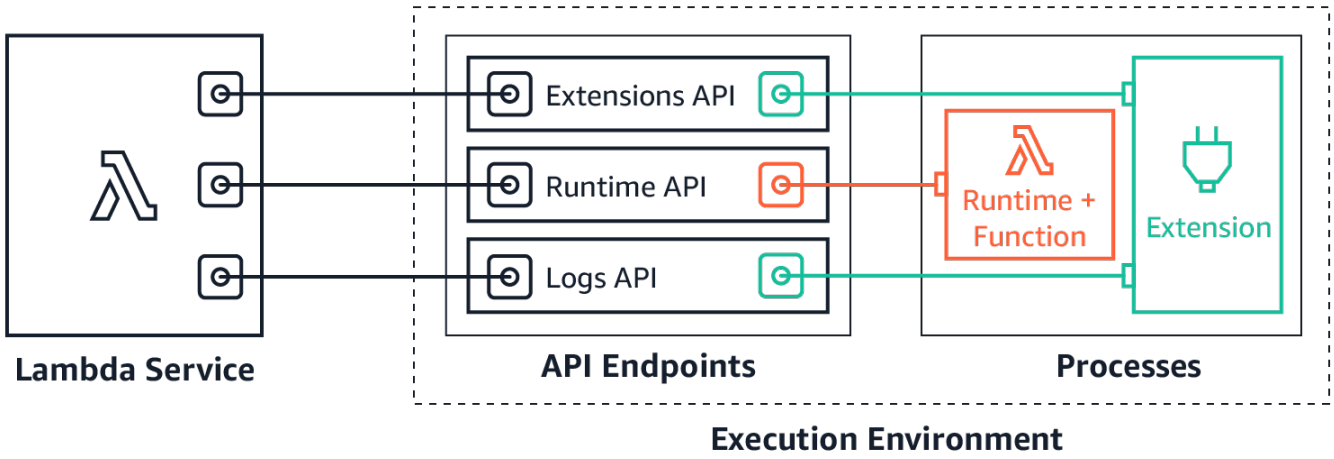
Menggunakan Lambda Logs API

> **Important**
>
> API Telemetri Lambda menggantikan API Lambda Logs. Meskipun API Log tetap berfungsi penuh, kami sarankan hanya menggunakan API Telemetri di masa mendatang. Anda dapat berlangganan ekstensi Anda ke aliran telemetri menggunakan API Telemetri atau API Log. Setelah berlangganan menggunakan salah satu API ini, setiap upaya untuk berlangganan menggunakan API lain akan menghasilkan kesalahan.

> **Instans Terkelola Lambda tidak mendukung API Log**
>
> Instans Terkelola Lambda tidak mendukung API Log. Jika Anda menggunakan fungsi Instans Terkelola, gunakan API Telemetri sebagai gantinya. API Telemetri menyediakan kemampuan yang disempurnakan untuk mengumpulkan dan memproses data telemetri dari Fungsi Acara Lambda Anda.

Lambda secara otomatis menangkap log runtime dan mengalirkannya ke Amazon. CloudWatch Aliran log ini berisi log yang dihasilkan oleh kode Fungsi Acara dan ekstensi Anda, dan juga log yang dihasilkan Lambda sebagai bagian dari pemanggilan Fungsi Peristiwa.

Ekstensi Lambda dapat menggunakan API Lambda Runtime Logs untuk berlangganan aliran log langsung dari lingkungan eksekusi Lambda. Lambda mengalirkan log ke ekstensi, dan ekstensi kemudian dapat memproses, memfilter, dan mengirim log ke tujuan yang diinginkan.



API Log memungkinkan ekstensi untuk berlangganan ke tiga aliran log yang berbeda:

- Fungsi mencatat bahwa Fungsi Peristiwa Lambda menghasilkan dan menulis ke `stdout` atau. `stderr`
- Log ekstensi yang dibuat kode ekstensi.
- Log platform Lambda, yang mencatat peristiwa dan kesalahan yang terkait dengan invokasi dan ekstensi.

> **Note**
>
> Lambda mengirimkan semua log ke CloudWatch, bahkan ketika ekstensi berlangganan ke satu atau beberapa aliran log.

Topik

- Berlangganan untuk menerima log
- Penggunaan memori
- Protokol tujuan
- Konfigurasi buffering
- Contoh berlangganan
- Kode sampel untuk API Log
- Referensi API Log
- Log pesan

Berlangganan untuk menerima log

Ekstensi Lambda dapat berlangganan untuk menerima log dengan mengirim permintaan langganan ke API Log.

Untuk berlangganan guna menerima log, Anda memerlukan pengidentifikasi ekstensi (`Lambda-Extension-Identifier`). Pertama, daftarkan ekstensi untuk menerima pengidentifikasi ekstensi. Kemudian berlangganan API Log selama inisialisasi. Setelah fase inisialisasi selesai, Lambda tidak memproses permintaan langganan.

> **Note**
>
> Berlangganan Log API adalah idempoten. Permintaan berlangganan duplikat tidak menghasilkan langganan duplikat.

Penggunaan memori

Penggunaan memori meningkat secara linear seiring bertambahnya jumlah pelanggan. Langganan menghabiskan sumber daya memori karena setiap langganan membuka buffer memori baru untuk menyimpan log. Untuk membantu mengoptimalkan penggunaan memori, Anda dapat menyesuaikan konfigurasi buffering. Penggunaan memori buffer dihitung untuk konsumsi memori keseluruhan dalam lingkungan eksekusi.

Protokol tujuan

Anda dapat memilih salah satu protokol berikut untuk menerima log:

1. HTTP (disarankan) – Lambda mengirimkan log ke titik akhir HTTP lokal (`http://sandbox.localdomain:${PORT}/${PATH}`) sebagai array catatan dalam format JSON. Parameter `$PATH` bersifat opsional. Perhatikan bahwa hanya HTTP yang didukung, bukan HTTPS. Anda dapat memilih untuk menerima log melalui PUT atau POST.
2. TCP – Lambda mengirimkan log ke port TCP dalam format JSON berbatas baris baru (NDJSON).

Kami menyarankan Anda untuk menggunakan HTTP alih-alih TCP. Dengan TCP, platform Lambda tidak dapat mengakui bahwa log dikirimkan ke lapisan aplikasi. Oleh karena itu, Anda mungkin kehilangan log jika ekstensi Anda mengalami crash. HTTP tidak berbagi batasan ini.

Kami juga menyarankan Anda untuk mengatur pendengar HTTP lokal atau port TCP sebelum berlangganan untuk menerima log. Selama penyetelan, perhatikan hal berikut ini:

- Lambda hanya mengirimkan log ke tujuan yang berada di lingkungan eksekusi.
- Lambda mengulang upaya untuk mengirim log (dengan pemunduran) jika tidak ada pendengar, atau jika permintaan POST atau PUT mengakibatkan kesalahan. Jika pelanggan log mengalami crahs, pelanggam terus menerima log setelah Lambda memulai ulang lingkungan eksekusi.
- Lambda mencadangkan port 9001. Tidak ada pembatasan atau rekomendasi nomor port lainnya.

Konfigurasi buffering

Lambda dapat mem-buffer log dan mengirimkannya kepada pelanggan. Anda dapat mengonfigurasi perilaku ini dalam permintaan langganan dengan menentukan bidang opsional berikut. Perhatikan bahwa Lambda menggunakan nilai default untuk setiap bidang yang tidak Anda tentukan.

- timeoutMs – Waktu maksimum (dalam milidetik) untuk mem-buffer batch. Default: 1.000. Minimum: 25. Maksimum: 30.000.
- maxBytes – Ukuran maksimum (dalam byte) log untuk buffer dalam memori. Default: 262.144. Minimum: 262.144. Maksimum: 1.048.576.
- maxItems – Jumlah maksimum peristiwa untuk buffer dalam memori. Default: 10.000. Minimum: 1.000. Maksimum: 10.000.

Selama konfigurasi buffering, perhatikan poin-poin berikut:

- Lambda membilas log jika salah satu aliran input tertutup, misalnya, jika waktu pengoperasian macet.
- Setiap pelanggan dapat menentukan konfigurasi buffering yang berbeda dalam permintaan langganan.
- Pertimbangkan ukuran buffer yang Anda butuhkan untuk membaca data. Perkirakan penerimaan muatan sebesar `2*maxBytes+metadata`, dengan `maxBytes` dikonfigurasi dalam permintaan berlangganan. Misalnya, Lambda menambahkan byte metadata berikut ke setiap catatan:

```
{
"time": "2020-08-20T12:31:32.123Z",
"type": "function",
"record": "Hello World"
}
```

- Jika pelanggan tidak dapat memproses log masuk dengan cukup cepat, Lambda mungkin akan menanggalkan log untuk menjaga pemanfaatan memori tetap terikat. Untuk menunjukkan jumlah catatan yang dihapus, Lambda mengirimkan log `platform.logsDropped`. Untuk informasi selengkapnya, lihat the section called “Lambda: Tidak semua log fungsi saya muncul”.

Contoh berlangganan

Contoh berikut menunjukkan permintaan untuk berlangganan ke platform dan log fungsi.

```
PUT http://${AWS_LAMBDA_RUNTIME_API}/2020-08-15/logs HTTP/1.1
{ "schemaVersion": "2020-08-15",
  "types": [
      "platform",
      "function"
    ],
  "buffering": {
      "maxItems": 1000,
      "maxBytes": 262144,
      "timeoutMs": 100
    },
  "destination": {
    "protocol": "HTTP",
    "URI": "http://sandbox.localdomain:8080/lambda_logs"
  }
}
```

Jika permintaan berhasil, pelanggan akan menerima respons keberhasilan HTTP 200.

```
HTTP/1.1 200 OK
"OK"
```

Kode sampel untuk API Log

Untuk kode contoh yang menunjukkan cara mengirim log ke tujuan kustom, lihat Menggunakan AWS Lambda ekstensi untuk mengirim log ke tujuan khusus di Blog AWS Komputasi.

Untuk contoh kode Python dan Go yang menunjukkan cara mengembangkan ekstensi Lambda dasar dan berlangganan API Log, lihat AWS Lambda Ekstensi pada repositori Sampel. AWS GitHub Untuk informasi lebih lanjut tentang membangun ekstensi Lambda, lihat the section called “API Ekstensi”.

Referensi API Log

Anda dapat mengambil titik akhir API Log dari variabel lingkungan `AWS_LAMBDA_RUNTIME_API`. Untuk mengirim permintaan API, gunakan prefiks `2020-08-15/` sebelum jalur API. Sebagai contoh:

```
http://${AWS_LAMBDA_RUNTIME_API}/2020-08-15/logs
```

Spesifikasi OpenAPI untuk permintaan langganan API Log, versi 2020-08-15, tersedia di sini: logs-api-request.zip

Langganan

Untuk berlangganan satu atau beberapa aliran log yang tersedia di lingkungan eksekusi Lambda, ekstensi akan mengirim permintaan API Berlangganan.

Jalur – `/logs`

Metode – PUT

Parameter tubuh

`destination` – Lihat the section called “Protokol tujuan”. Wajib: ya. Jenis: string.

`buffering` – Lihat the section called “Konfigurasi buffering”. Wajib: tidak. Jenis: string.

`types` – Array jenis log yang akan diterima. Wajib: ya. Jenis: array string. Nilai valid: "platform", "function", "extension".

`schemaVersion` – Wajib: tidak. Nilai default: "2020-08-15". Atur ke "2021-03-18" untuk ekstensi untuk menerima pesan `platform.runtimeDone`.

Parameter respons

Spesifikasi OpenAPI untuk respons langganan versi 2020-08-15 tersedia untuk protokol HTTP dan TCP:

- HTTP: logs-api-http-response.zip
- TCP: logs-api-tcp-response.zip

Kode respons

- 200 – Permintaan berhasil diselesaikan
- 202 – Permintaan diterima. Respons terhadap permintaan berlangganan selama pengujian lokal.
- 4XX – Permintaan Buruk
- 500 – Kesalahan Layanan

Jika permintaan berhasil, pelanggan akan menerima respons keberhasilan HTTP 200.

```
HTTP/1.1 200 OK
"OK"
```

Jika permintaan gagal, pelanggan akan menerima respons kesalahan. Sebagai contoh:

```
HTTP/1.1 400 OK
{
    "errorType": "Logs.ValidationError",
    "errorMessage": URI port is not provided; types should not be empty"
}
```

Log pesan

API Log memungkinkan ekstensi untuk berlangganan ke tiga aliran log yang berbeda:

- Fungsi — Log yang dihasilkan dan ditulis oleh Fungsi Peristiwa Lambda ke `stdout` atau. `stderr`
- Ekstenso – Log yang dibuat kode ekstensi.
- Platform –Log yang dibuat platform runtime, yang mencatat peristiwa dan kesalahan yang terkait dengan invokasi dan ekstensi.

Topik

- Log fungsi
- Log ekstensi:
- Log platform

Log fungsi

Fungsi Peristiwa Lambda dan ekstensi internal menghasilkan log fungsi dan menuliskannya ke `stdout` atau. `stderr`

Contoh berikut menunjukkan format pesan log fungsi. {“time”: “2020-08-20 “, T12:31:32.123Z “type”: “function”, “record”: “ERROR ditemui. Stack trace:\n\my-function (line 10)\n" }

Log ekstensi:

Ekstensi dapat menghasilkan log ekstensi. Format log sama seperti untuk log fungsi.

Log platform

Lambda menghasilkan pesan log untuk peristiwa platform seperti `platform.start`, `platform.end`, dan `platform.fault`.

Atau, Anda dapat berlangganan ke versi 2021-03-18skema API Log, yang mencakup pesan log `platform.runtimeDone`.

Contoh pesan log platform

Contoh berikut menunjukkan log awal platform dan akhir platform. Log ini menunjukkan waktu mulai invokasi dan waktu berakhir invokasi untuk pemanggilan yang ditentukan oleh requestId.

```
{
    "time": "2020-08-20T12:31:32.123Z",
    "type": "platform.start",
    "record": {"requestId": "6f7f0961f83442118a7af6fe80b88d56"}
}
{
    "time": "2020-08-20T12:31:32.123Z",
    "type": "platform.end",
    "record": {"requestId": "6f7f0961f83442118a7af6fe80b88d56"}
}
```

Pesan RuntimeDone log platform.init menunjukkan status `Runtime init` sub-fase, yang merupakan bagian dari fase lifecyle Init. `Runtime init`Kapan berhasil, runtime mengirimkan permintaan API `/next` runtime (untuk tipe `on-demand` dan `provisioned-concurrency` inisialisasi) atau `restore/next` (untuk tipe `snap-start` inisialisasi). Contoh berikut menunjukkan pesan RuntimeDone log platform.init yang berhasil untuk jenis inisialisasi. `snap-start`

```
{
  "time":"2022-07-17T18:41:57.083Z",
  "type":"platform.initRuntimeDone",
  "record":{
      "initializationType":"snap-start",
      "status":"success"
  }
}
```

Pesan log Platform.initReport menunjukkan berapa lama `Init` fase berlangsung dan berapa milidetik Anda ditagih selama fase ini. Ketika jenis inisialisasi`provisioned-concurrency`, Lambda mengirimkan pesan ini selama pemanggilan. Saat jenis inisialisasi`snap-start`, Lambda mengirimkan pesan ini setelah memulihkan snapshot. Contoh berikut menunjukkan pesan log Platform.initReport untuk jenis inisialisasi. `snap-start`

```
{
```

```
  "time":"2022-07-17T18:41:57.083Z",
  "type":"platform.initReport",
  "record":{
      "initializationType":"snap-start",
      "metrics":{
          "durationMs":731.79,
          "billedDurationMs":732
          }
  }
}
```

Log laporan platform menyertakan metrik tentang invokasi yang ditentukan oleh requestId. Bidang `initDurationMs` termasuk dalam log hanya jika invokasi termasuk mulai awal. Jika AWS X-Ray penelusuran aktif, log menyertakan X-Ray metadata. Contoh berikut menunjukkan log laporan platform untuk invokasi yang mencakup mulai awal.

```
{
    "time": "2020-08-20T12:31:32.123Z",
    "type": "platform.report",
    "record": {"requestId": "6f7f0961f83442118a7af6fe80b88d56",
        "metrics": {"durationMs": 101.51,
            "billedDurationMs": 300,
            "memorySizeMB": 512,
            "maxMemoryUsedMB": 33,
            "initDurationMs": 116.67
        }
    }
}
```

Log kesalahan platform menangkap waktu pengoperasian atau kesalahan lingkungan eksekusi. Contoh berikut ini menunjukkan pesan log kesalahan platform.

```
{
    "time": "2020-08-20T12:31:32.123Z",
    "type": "platform.fault",
    "record": "RequestId: d783b35e-a91d-4251-af17-035953428a2c Process exited before
 completing request"
}
```

> **Note**
>
> AWS saat ini menerapkan perubahan pada layanan Lambda. Karena perubahan ini, Anda mungkin melihat perbedaan kecil antara struktur dan konten pesan log sistem dan segmen pelacakan yang dipancarkan oleh Fungsi Peristiwa Lambda yang berbeda di Anda. Akun AWS
> Salah satu output log yang terpengaruh oleh perubahan ini adalah `"record"` bidang log kesalahan platform. Contoh berikut menunjukkan `"record"` bidang ilustrasi dalam format lama dan baru. Gaya baru log kesalahan berisi pesan yang lebih ringkas
> Perubahan ini akan diterapkan selama beberapa minggu mendatang, dan semua fungsi di semua Wilayah AWS kecuali China GovCloud dan wilayah akan bertransisi untuk menggunakan pesan log format baru dan segmen pelacakan.

Example catatan log kesalahan platform (gaya lama)

```
"record":"RequestId: ...\tError: Runtime exited with error: exit status
 255\nRuntime.ExitError"
```

Example catatan log kesalahan platform (gaya baru)

```
"record":"RequestId: ... Status: error\tErrorType: Runtime.ExitError"
```

Lambda membuat log ekstensi platform ketika ekstensi mendaftar dengan API ekstensi. Contoh berikut ini menunjukkan pesan ekstensi platform.

```
{
    "time": "2020-08-20T12:31:32.123Z",
    "type": "platform.extension",
    "record": {"name": "Foo.bar",
        "state": "Ready",
        "events": ["INVOKE", "SHUTDOWN"]
     }
}
```

Lambda menghasilkan log langganan log platform saat ekstensi berlangganan API log. Contoh berikut menunjukkan pesan langganan Log.

```
{
    "time": "2020-08-20T12:31:32.123Z",
    "type": "platform.logsSubscription",
    "record": {"name": "Foo.bar",
        "state": "Subscribed",
        "types": ["function", "platform"],
    }
}
```

Lambda menghasilkan log yang dihapus log platform ketika ekstensi tidak dapat memproses jumlah log yang diterimanya. Contoh berikut menunjukkan pesan log `platform.logsDropped`.

```
{
    "time": "2020-08-20T12:31:32.123Z",
    "type": "platform.logsDropped",
    "record": {"reason": "Consumer seems to have fallen behind as it has not
 acknowledged receipt of logs.",
        "droppedRecords": 123,
        "droppedBytes" 12345
    }
}
```

Pesan log Platform.RestoRestart menunjukkan waktu `Restore` fase dimulai (`snap-start`hanya jenis inisialisasi). Contoh:

```
{
  "time":"2022-07-17T18:43:44.782Z",
  "type":"platform.restoreStart",
  "record":{}
}
```

Pesan log Platform.RestoreReport menunjukkan berapa lama `Restore` fase berlangsung dan berapa milidetik Anda ditagih selama fase ini (hanya jenis inisialisasi). `snap-start` Contoh:

```
{
  "time":"2022-07-17T18:43:45.936Z",
  "type":"platform.restoreReport",
  "record":{
      "metrics":{
          "durationMs":70.87,
          "billedDurationMs":13
```

```
      }
  }
}
```

Pesan platform **`RuntimeDone`**

Jika Anda menyetel versi skema ke “2021-03-18" dalam permintaan berlangganan, Lambda mengirim `platform.runtimeDone` pesan setelah pemanggilan Fungsi Acara selesai dengan sukses atau dengan kesalahan. Ekstensi dapat menggunakan pesan ini untuk menghentikan semua koleksi telemetri untuk pemanggilan Fungsi Acara ini.

Spesifikasi OpenAPI untuk tipe peristiwa Log dalam versi skema 2021-03-18 tersedia di sini: schema-2021-03-18.zip

Lambda menghasilkan pesan log `platform.runtimeDone` ketika runtime mengirimkan permintaan API runtime `Next` atau `Error`. Log `platform.runtimeDone` memberi tahu konsumen API Log bahwa fungsi invokasi selesai. Ekstensi dapat menggunakan informasi ini untuk memutuskan kapan harus mengirim semua telemetri yang dikumpulkan selama invokasi tersebut.

Contoh

Lambda mengirimkan pesan `platform.runtimeDone` setelah runtime mengirimkan permintaan NEXT ketika fungsi invokasi selesai. Contoh berikut menunjukkan pesan untuk masing-masing nilai status: sukses, kegagalan, dan waktu habis.

Example Contoh pesan sukses

```
{
    "time": "2021-02-04T20:00:05.123Z",
    "type": "platform.runtimeDone",
    "record": {
       "requestId":"6f7f0961f83442118a7af6fe80b88",
       "status": "success"
    }
}
```

Example Contoh pesan kegagalan

```
{
   "time": "2021-02-04T20:00:05.123Z",
```

```
   "type": "platform.runtimeDone",
   "record": {
      "requestId":"6f7f0961f83442118a7af6fe80b88",
      "status": "failure"
   }
}
```

Example Contoh pesan waktu habis

```
{
   "time": "2021-02-04T20:00:05.123Z",
   "type": "platform.runtimeDone",
   "record": {
      "requestId":"6f7f0961f83442118a7af6fe80b88",
      "status": "timeout"
  }
}
```

Example Contoh RuntimeDone pesan platform.restore (hanya tipe inisialisasi **`snap-start`**)

Pesan RuntimeDone log platform.restore menunjukkan apakah `Restore` fase berhasil atau tidak. Lambda mengirimkan pesan ini saat runtime mengirimkan permintaan API `restore/next` runtime. Ada tiga kemungkinan status: sukses, gagal, dan batas waktu. Contoh berikut menunjukkan pesan log platform.restore RuntimeDone yang berhasil.

```
{
  "time":"2022-07-17T18:43:45.936Z",
  "type":"platform.restoreRuntimeDone",
  "record":{
      "status":"success"
  }
}
```

Memecahkan masalah di Lambda

Topik berikut memberikan saran pemecahan masalah untuk kesalahan dan masalah yang mungkin Anda temui saat menggunakan API, konsol, dan peralatan Lambda. Jika Anda menemukan masalah yang tidak tercantum di sini, Anda dapat menggunakan tombol Umpan Balik di halaman ini untuk melaporkannya.

Untuk saran pemecahan masalah selengkapnya dan jawaban atas pertanyaan dukungan umum, kunjungi Pusat Pengetahuan AWS.

Untuk informasi selengkapnya tentang debugging dan pemecahan masalah aplikasi Lambda, lihat Debugging di Serverless Land.

Topik

- Memecahkan masalah konfigurasi di Lambda
- Menyelesaikan masalah deployment di Lambda
- Menyelesaikan masalah invokasi di Lambda
- Menyelesaikan masalah eksekusi di Lambda
- Memecahkan masalah pemetaan sumber acara di Lambda
- Menyelesaikan masalah jaringan di Lambda

Memecahkan masalah konfigurasi di Lambda

Pengaturan konfigurasi fungsi Anda dapat berdampak pada kinerja dan perilaku Fungsi Peristiwa Lambda secara keseluruhan. Ini mungkin tidak menyebabkan kesalahan fungsi yang sebenarnya, tetapi dapat menyebabkan batas waktu dan hasil yang tidak terduga.

Topik berikut memberikan saran pemecahan masalah untuk masalah umum yang mungkin Anda temui terkait dengan pengaturan konfigurasi fungsi Lambda.

Topik

- Konfigurasi memori
- CPU-bound konfigurasi
- Timeout
- Kebocoran memori di antara pemanggilan
- Hasil asinkron dikembalikan ke pemanggilan selanjutnya

Konfigurasi memori

Anda dapat mengonfigurasi fungsi Lambda untuk menggunakan antara 128 MB dan 10.240 MB memori. Secara default, fungsi apa pun yang dibuat di konsol diberi jumlah memori terkecil. Banyak fungsi Lambda berkinerja pada pengaturan terendah ini. Namun, jika Anda mengimpor pustaka kode besar atau menyelesaikan tugas intensif memori, 128 MB tidak cukup.

Jika fungsi Anda berjalan jauh lebih lambat dari yang diharapkan, langkah pertama adalah meningkatkan pengaturan memori. Untuk fungsi terikat memori, ini menyelesaikan kemacetan dan mungkin meningkatkan kinerja fungsi Anda.

CPU-bound konfigurasi

Untuk operasi komputasi intensif, jika fungsi Anda mengalami kinerja yang lebih lambat dari perkiraan, ini mungkin disebabkan oleh fungsi Anda. CPU-bound Dalam hal ini, kapasitas komputasi fungsi tidak dapat mengimbangi pekerjaan.

Meskipun Lambda tidak memungkinkan Anda untuk memodifikasi konfigurasi CPU secara langsung, CPU secara tidak langsung dikontrol melalui pengaturan memori. Layanan Lambda secara proporsional mengalokasikan lebih banyak CPU virtual saat Anda mengalokasikan lebih banyak memori. Pada memori 1, 8 GB, fungsi Lambda memiliki seluruh vCPU yang dialokasikan, dan di atas level ini ia memiliki akses ke lebih dari satu inti vCPU. Pada 10.240 MB, ia memiliki 6 vCPU yang tersedia. Dengan kata lain, Anda dapat meningkatkan kinerja dengan meningkatkan alokasi memori, bahkan jika fungsi tersebut tidak menggunakan semua memori.

Timeout

Timeout untuk fungsi Lambda dapat diatur antara 1 dan 900 detik (15 menit). Untuk fungsi yang menggunakan Instans AWS Lambda Terkelola, pemanggilan asinkron dan pemanggilan pemetaan sumber peristiwa (kecuali Amazon MQ dan Amazon DocumentDB) mendukung nilai maksimum yang diizinkan 5.400 detik (90 menit). Secara default, konsol Lambda menyetel ini menjadi 3 detik. Nilai batas waktu adalah katup pengaman yang memastikan fungsi tidak berjalan tanpa batas waktu. Setelah nilai batas waktu tercapai, Lambda menghentikan pemanggilan fungsi.

Jika nilai batas waktu ditetapkan mendekati durasi rata-rata suatu fungsi, ini meningkatkan risiko bahwa fungsi tersebut habis secara tak terduga. Durasi suatu fungsi dapat bervariasi berdasarkan jumlah transfer dan pemrosesan data, dan latensi layanan apa pun yang berinteraksi dengan fungsi tersebut. Penyebab umum batas waktu meliputi:

- Saat mengunduh data dari bucket S3 atau penyimpanan data lainnya, unduhan lebih besar atau membutuhkan waktu lebih lama dari rata-rata.
- Fungsi membuat permintaan ke layanan lain, yang membutuhkan waktu lebih lama untuk merespons.
- Parameter yang disediakan untuk suatu fungsi membutuhkan lebih banyak kompleksitas komputasi dalam fungsi, yang menyebabkan pemanggilan memakan waktu lebih lama.

Saat menguji aplikasi Anda, pastikan pengujian Anda secara akurat mencerminkan ukuran dan kuantitas data, serta nilai parameter yang realistis. Yang penting, gunakan kumpulan data di batas atas dari apa yang diharapkan secara wajar untuk beban kerja Anda.

Selain itu, terapkan batas batas atas dalam beban kerja Anda di mana pun praktis. Dalam contoh ini, aplikasi dapat menggunakan batas ukuran maksimum untuk setiap jenis file. Anda kemudian dapat menguji kinerja aplikasi Anda untuk berbagai ukuran file yang diharapkan, hingga dan termasuk batas maksimum.

Kebocoran memori di antara pemanggilan

Variabel global dan objek yang disimpan dalam fase INIT dari pemanggilan Lambda mempertahankan statusnya di antara pemanggilan hangat. Mereka sepenuhnya diatur ulang hanya ketika lingkungan eksekusi dijalankan untuk pertama kalinya (juga dikenal sebagai “start dingin”). Setiap variabel yang disimpan dalam handler dihancurkan ketika handler keluar. Praktik terbaik adalah menggunakan fase INIT untuk mengatur koneksi database, memuat pustaka, membuat cache, dan memuat aset yang tidak dapat diubah.

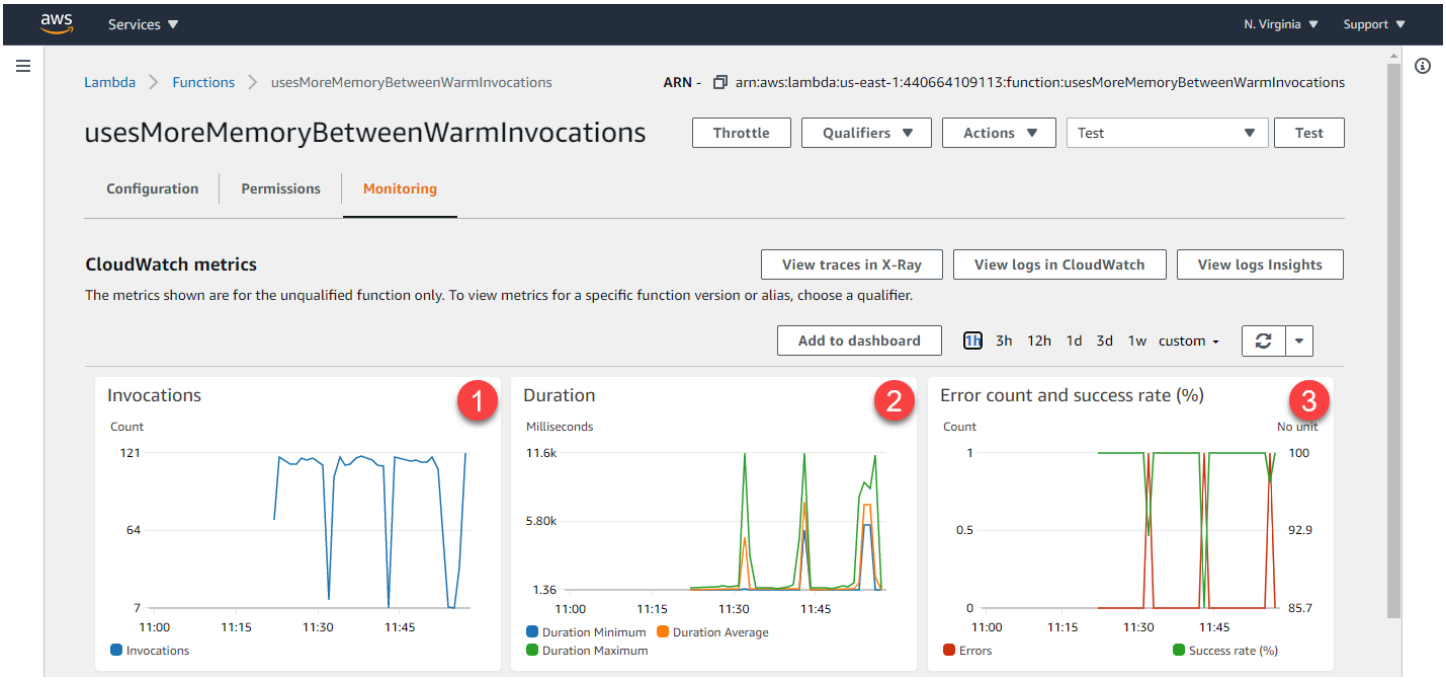
Saat Anda menggunakan pustaka pihak ketiga di beberapa pemanggilan di lingkungan eksekusi yang sama, periksa dokumentasi mereka untuk penggunaan di lingkungan komputasi tanpa server. Beberapa koneksi database dan pustaka logging mungkin menyimpan hasil pemanggilan menengah dan data lainnya. Hal ini menyebabkan penggunaan memori perpustakaan ini tumbuh dengan pemanggilan hangat berikutnya. Jika ini masalahnya, Anda mungkin menemukan fungsi Lambda kehabisan memori, bahkan jika kode kustom Anda membuang variabel dengan benar.

Masalah ini memengaruhi pemanggilan yang terjadi di lingkungan eksekusi yang hangat. Misalnya, kode berikut membuat kebocoran memori antara pemanggilan. Fungsi Lambda mengkonsumsi memori tambahan dengan setiap pemanggilan dengan meningkatkan ukuran array global:

```
let a = []

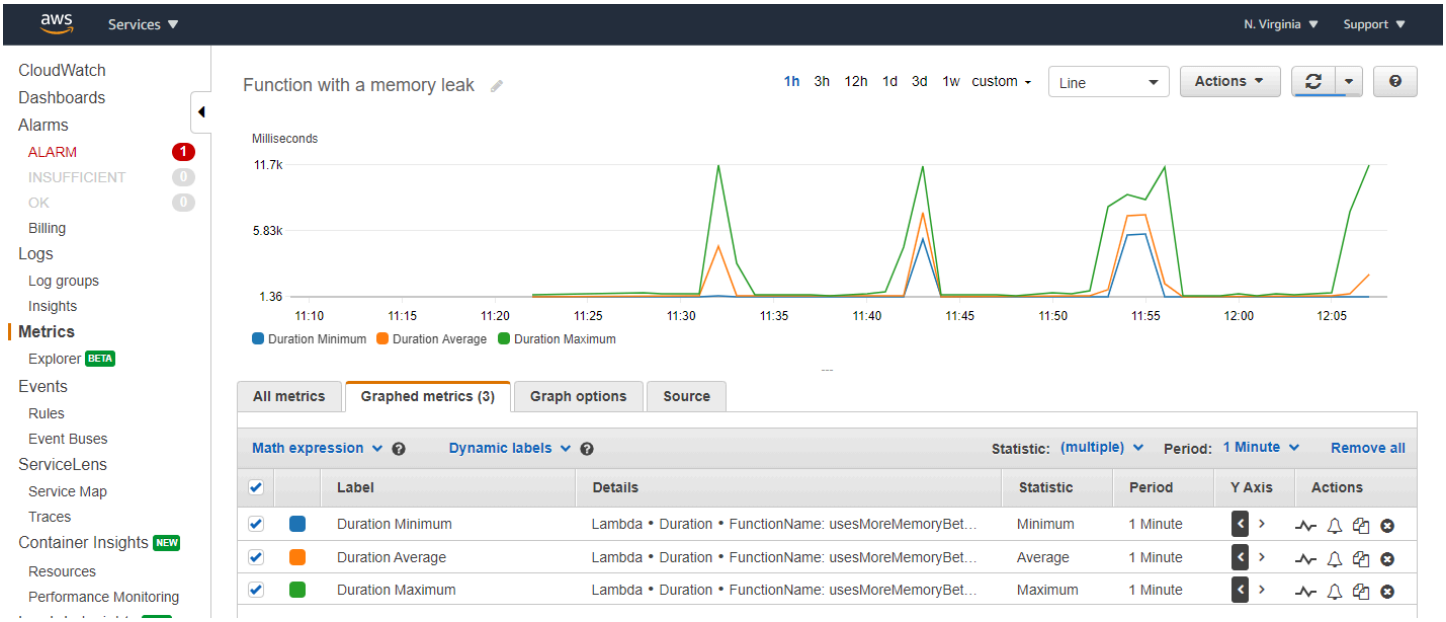
exports.handler = async (event) => {
    a.push(Array(100000).fill(1))
}
```

Dikonfigurasi dengan memori 128 MB, setelah menjalankan fungsi ini 1000 kali, tab Pemantauan fungsi Lambda menunjukkan perubahan khas dalam pemanggilan, durasi, dan jumlah kesalahan saat kebocoran memori terjadi:



1. Doa — Tingkat transaksi yang stabil terganggu secara berkala karena pemanggilan membutuhkan waktu lebih lama untuk diselesaikan. Selama kondisi tunak, kebocoran memori tidak menghabiskan semua memori yang dialokasikan fungsi. Ketika kinerja menurun, sistem operasi melakukan paging penyimpanan lokal untuk mengakomodasi memori yang tumbuh yang dibutuhkan oleh fungsi, yang menghasilkan lebih sedikit transaksi yang diselesaikan.
2. Durasi — Sebelum fungsi kehabisan memori, ia menyelesaikan pemanggilan dengan kecepatan milidetik dua digit yang stabil. Saat paging terjadi, durasinya membutuhkan urutan besarnya lebih lama.
3. Jumlah kesalahan — Karena kebocoran memori melebihi memori yang dialokasikan, akhirnya kesalahan fungsi karena perhitungan melebihi batas waktu, atau lingkungan eksekusi menghentikan fungsi.

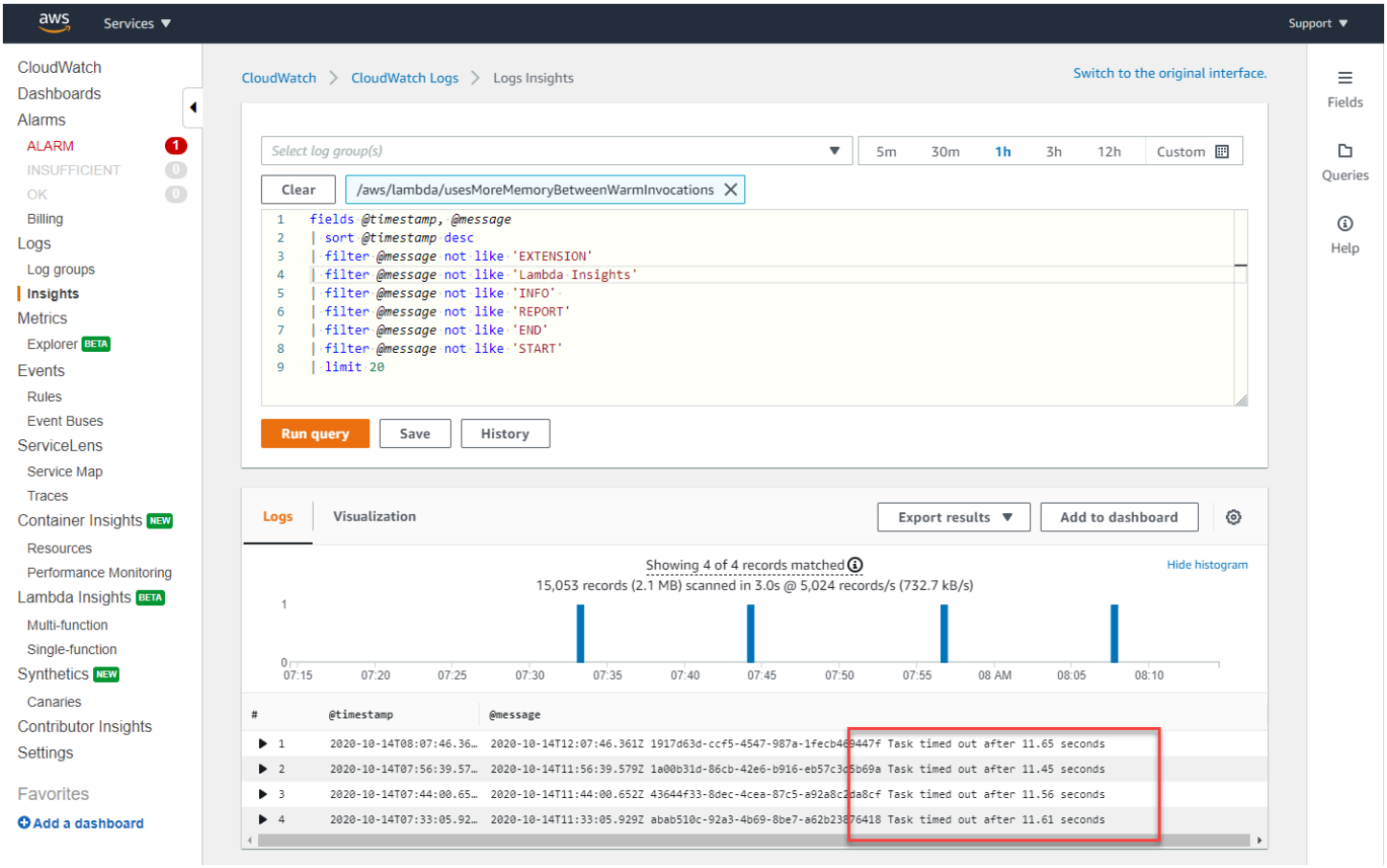
Setelah kesalahan, Lambda memulai ulang lingkungan eksekusi, yang menjelaskan mengapa ketiga grafik menunjukkan pengembalian ke keadaan semula. Memperluas CloudWatch metrik untuk durasi memberikan detail lebih lanjut untuk statistik durasi minimum, maksimum, dan rata-rata:



Untuk menemukan kesalahan yang dihasilkan di 1000 pemanggilan, Anda dapat menggunakan bahasa kueri CloudWatch Insights. Kueri berikut mengecualikan log informasi untuk melaporkan hanya kesalahan:

```
fields @timestamp, @message
| sort @timestamp desc
| filter @message not like 'EXTENSION'
| filter @message not like 'Lambda Insights'
| filter @message not like 'INFO'
| filter @message not like 'REPORT'
| filter @message not like 'END'
| filter @message not like 'START'
```

Ketika dijalankan terhadap grup log untuk fungsi ini, ini menunjukkan bahwa batas waktu bertanggung jawab atas kesalahan periodik:



Hasil asinkron dikembalikan ke pemanggilan selanjutnya

Untuk kode fungsi yang menggunakan pola asinkron, hasil callback dari satu pemanggilan dapat dikembalikan dalam pemanggilan future. Contoh ini menggunakan Node.js, tetapi logika yang sama dapat diterapkan ke runtime lain menggunakan pola asinkron. Fungsi ini menggunakan sintaks callback tradisional di. JavaScript Ini memanggil fungsi asinkron dengan penghitung inkremental yang melacak jumlah pemanggilan:

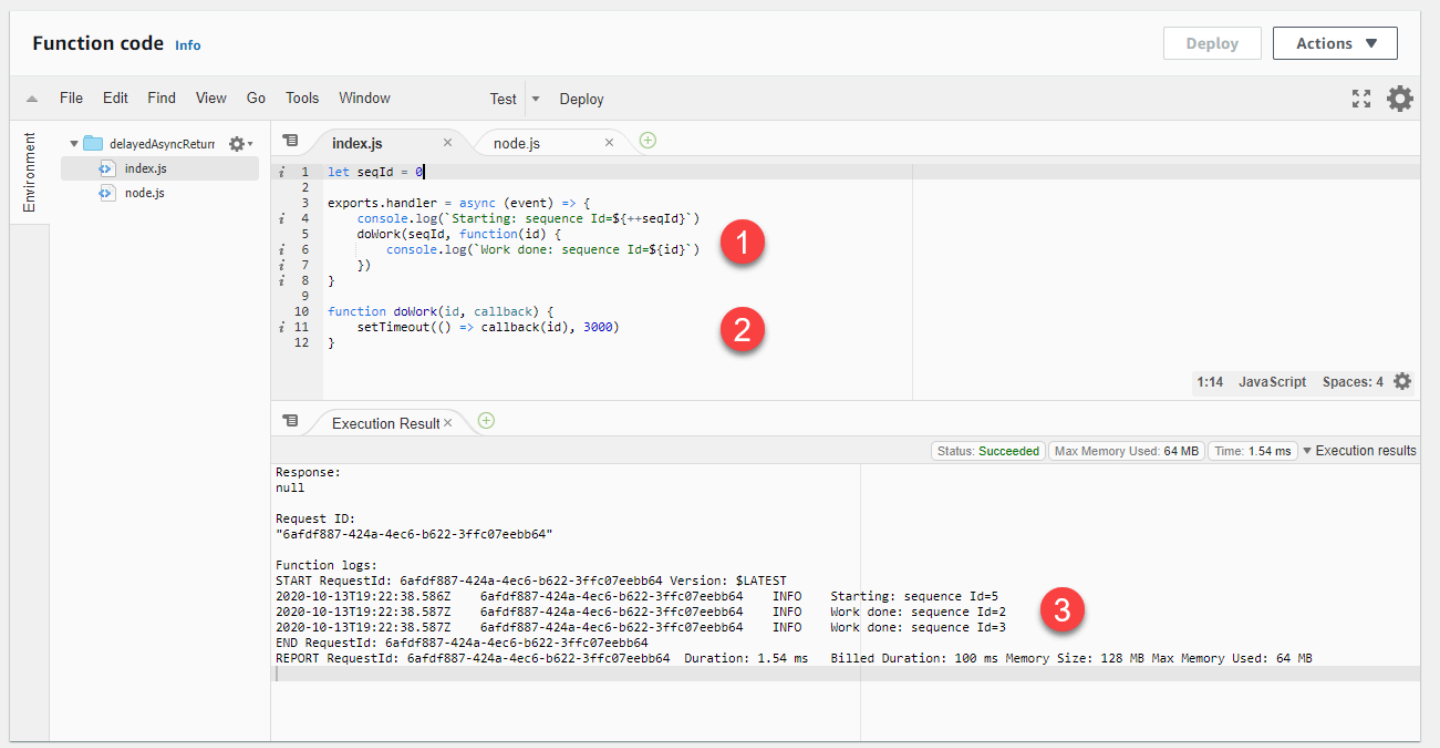
```
let seqId = 0

exports.handler = async (event, context) => {
    console.log(`Starting: sequence Id=${++seqId}`)
    doWork(seqId, function(id) {
        console.log(`Work done: sequence Id=${id}`)
    })
}

function doWork(id, callback) {
    setTimeout(() => callback(id), 3000)
```

```
}
```

Ketika dipanggil beberapa kali berturut-turut, hasil callback terjadi dalam pemanggilan berikutnya:



1. Kode memanggil `doWork` fungsi, menyediakan fungsi callback sebagai parameter terakhir.
2. `doWork`Fungsi ini membutuhkan beberapa periode waktu untuk diselesaikan sebelum memanggil panggilan balik.
3. Pencatatan fungsi menunjukkan bahwa pemanggilan berakhir sebelum `doWork` fungsi selesai dieksekusi. Selain itu, setelah memulai iterasi, callback dari iterasi sebelumnya sedang diproses, seperti yang ditunjukkan dalam log.

Dalam JavaScript, panggilan balik asinkron ditangani dengan loop peristiwa. Runtime lain menggunakan mekanisme yang berbeda untuk menangani konkurensi. Ketika lingkungan eksekusi fungsi berakhir, Lambda membekukan lingkungan hingga pemanggilan berikutnya.

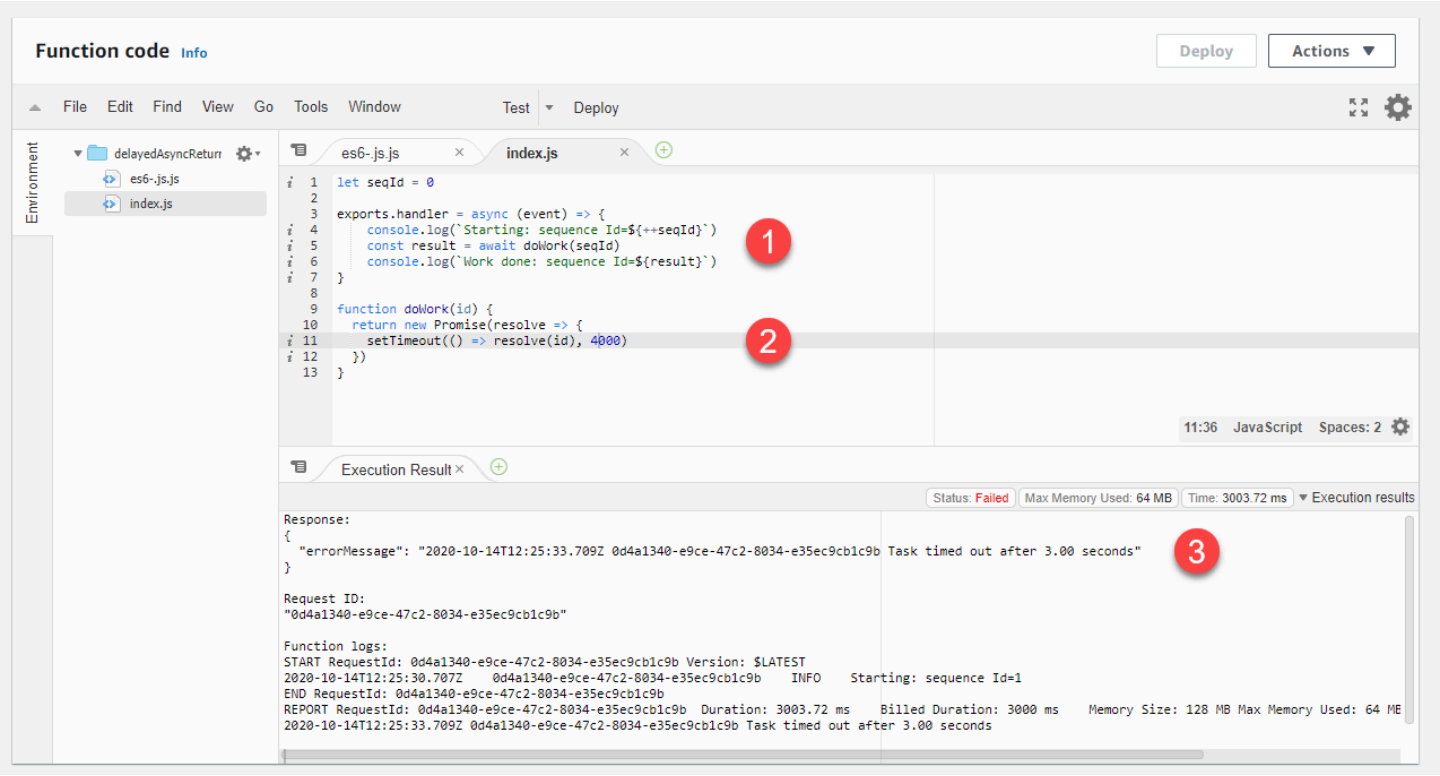
Setelah dilanjutkan, JavaScript lanjutkan pemrosesan loop peristiwa, yang dalam hal ini menyertakan panggilan balik asinkron dari pemanggilan sebelumnya. Tanpa konteks ini, dapat muncul bahwa fungsi menjalankan kode tanpa alasan, dan mengembalikan data arbitrer. Faktanya, ini benar-benar artefak tentang bagaimana konkurensi runtime dan lingkungan eksekusi berinteraksi.

Ini menciptakan potensi data pribadi dari pemanggilan sebelumnya untuk muncul dalam pemanggilan berikutnya. Ada dua cara untuk mencegah atau mendeteksi perilaku ini. Pertama, JavaScript sediakan kata kunci async dan await untuk menyederhanakan pengembangan asinkron dan juga memaksa eksekusi kode untuk menunggu panggilan asinkron selesai. Fungsi di atas dapat ditulis ulang menggunakan pendekatan ini sebagai berikut:

```
let seqId = 0
exports.handler = async (event) => {
    console.log(`Starting: sequence Id=${++seqId}`)
    const result = await doWork(seqId)
    console.log(`Work done: sequence Id=${result}`)
}

function doWork(id) {
  return new Promise(resolve => {
    setTimeout(() => resolve(id), 4000)
  })
}
```

Menggunakan sintaks ini mencegah handler keluar sebelum fungsi asinkron selesai. Dalam kasus ini, jika callback membutuhkan waktu lebih lama dari batas waktu fungsi Lambda, fungsi akan menampilkan kesalahan, alih-alih mengembalikan hasil callback dalam pemanggilan selanjutnya:



1. Kode memanggil `doWork` fungsi asinkron menggunakan kata kunci await di handler.
2. `doWork`Fungsi ini membutuhkan beberapa periode waktu untuk menyelesaikan sebelum menyelesaikan janji.
3. Fungsi waktu habis karena `doWork` membutuhkan waktu lebih lama dari batas waktu yang diizinkan dan hasil panggilan balik tidak dikembalikan dalam pemanggilan selanjutnya.

Umumnya, Anda harus memastikan setiap proses latar belakang atau callback dalam kode selesai sebelum kode keluar. Jika hal ini tidak memungkinkan dalam kasus penggunaan Anda, Anda dapat menggunakan pengenal untuk memastikan bahwa callback milik pemanggilan saat ini. Untuk melakukan ini, Anda dapat menggunakan aws RequestId yang disediakan oleh objek konteks. Dengan meneruskan nilai ini ke callback asinkron, Anda dapat membandingkan nilai yang diteruskan dengan nilai saat ini untuk mendeteksi apakah panggilan balik berasal dari pemanggilan lain:

```
let currentContext

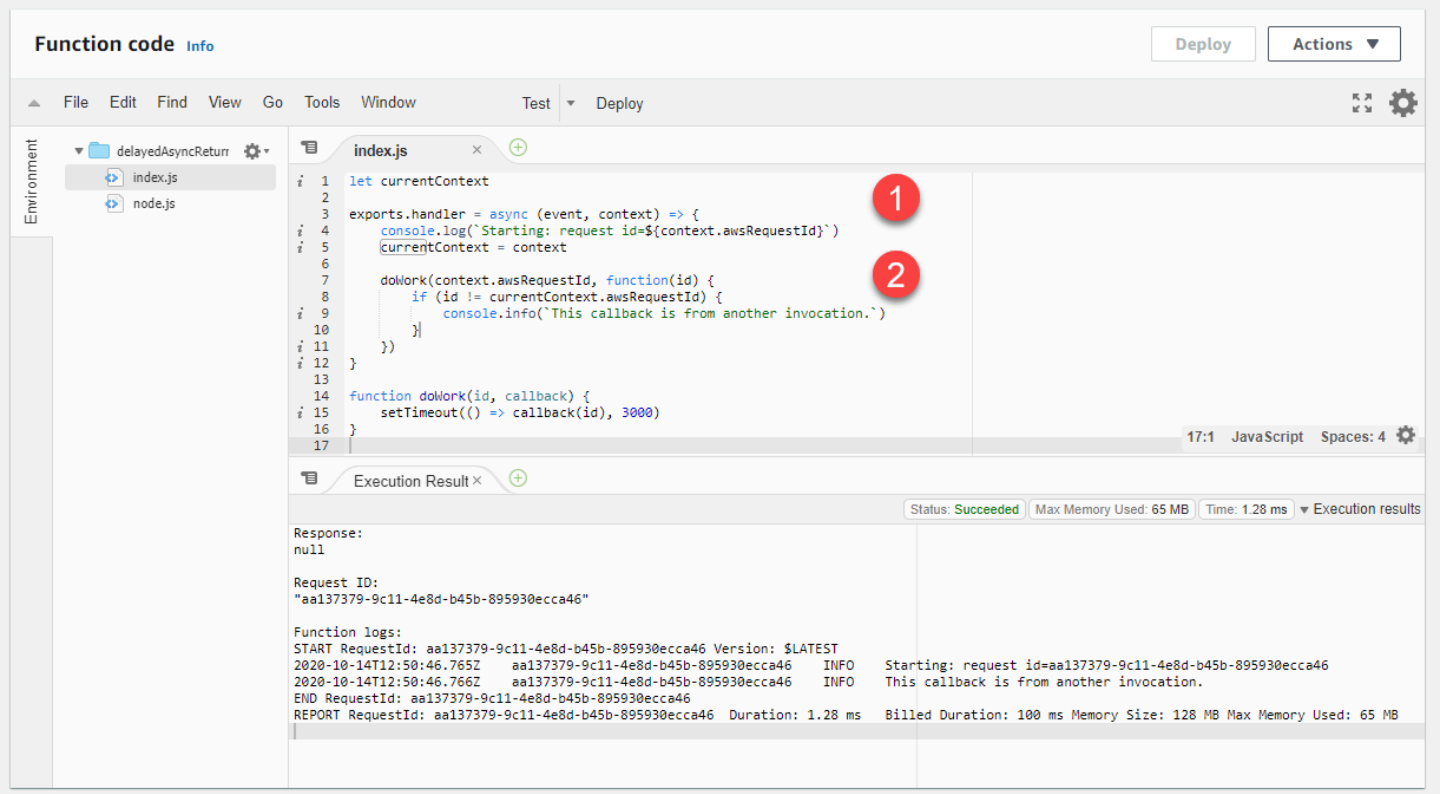
exports.handler = async (event, context) => {
    console.log(`Starting: request id=$\{context.awsRequestId}`)
    currentContext = context

    doWork(context.awsRequestId, function(id) {
        if (id != currentContext.awsRequestId) {
            console.info(`This callback is from another invocation.`)
        }
    })

}

function doWork(id, callback) {
    setTimeout(() => callback(id), 3000)

}
```



1. Penangan fungsi Lambda mengambil parameter konteks, yang menyediakan akses ke ID permintaan pemanggilan unik.
2. `awsRequestId`Ini diteruskan ke fungsi DoWork. Dalam callback, ID dibandingkan dengan `awsRequestId` pemanggilan saat ini. Jika nilai-nilai ini berbeda, kode dapat mengambil tindakan yang sesuai.

Menyelesaikan masalah deployment di Lambda

Saat Anda memperbarui Fungsi Peristiwa Lambda, Lambda menerapkan perubahan dengan meluncurkan instance baru fungsi dengan kode atau pengaturan yang diperbarui. Kesalahan deployment menghambat penggunaan versi baru dan dapat disebabkan oleh masalah dengan paket deployment, kode, izin, atau alat Anda.

Saat Anda menerapkan pembaruan ke fungsi Anda secara langsung dengan API Lambda atau dengan klien seperti, Anda dapat melihat AWS CLI kesalahan dari Lambda secara langsung di output. Jika Anda menggunakan layanan seperti AWS CloudFormation, AWS CodeDeploy, atau AWS CodePipeline, cari respons dari Lambda di log atau aliran peristiwa untuk layanan tersebut.

Topik berikut memberikan saran pemecahan masalah untuk kesalahan dan masalah yang mungkin Anda temui saat menggunakan API, konsol, dan peralatan Lambda. Jika Anda menemukan masalah yang tidak tercantum di sini, Anda dapat menggunakan tombol Umpan Balik di halaman ini untuk melaporkannya.

Untuk saran pemecahan masalah selengkapnya dan jawaban atas pertanyaan dukungan umum, kunjungi Pusat Pengetahuan AWS.

Untuk informasi selengkapnya tentang debugging dan pemecahan masalah aplikasi Lambda, lihat Debugging di Tanah Tanpa Server.

Topik

- Umum: Izin ditolak atau Tidak dapat memuat file tersebut
- Umum: Terjadi kesalahan saat memanggil UpdateFunctionCode
- Amazon S3: Kode Kesalahan. PermanentRedirect
- Umum: Tidak dapat menemukan, tidak dapat memuat, tidak dapat mengimpor, kelas tidak ditemukan, tidak ada file atau direktori tersebut
- Umum: Metode handler tidak didefinisikan
- Umum: Batas penyimpanan kode Lambda terlampaui
- Lambda: Konversi lapisan gagal
- Lambda: atau InvalidParameterValueException RequestEntityTooLargeException
- Lambda: InvalidParameterValueException
- Lambda: Konkurensi dan kuota memori
- Lambda: Konfigurasi alias tidak valid untuk konkurensi yang disediakan

Umum: Izin ditolak atau Tidak dapat memuat file tersebut

Kesalahan: EACCES: izin ditolak, buka var/task '/ /index.js '

Kesalahan: tidak dapat memuat file tersebut -- fungsi

Kesalahan: [Errno 13] Izin ditolak: '/ var/task /function.py '

Runtime Lambda membutuhkan izin untuk membaca file dalam paket deployment Anda. Dalam notasi oktal izin Linux, Lambda membutuhkan 644 izin untuk file yang tidak dapat dieksekusi (rw-r - r--) dan 755 izin (rwxr-xr-x) untuk direktori dan file yang dapat dieksekusi.

Di Linux dan macOS, gunakan `chmod` perintah untuk mengubah izin file pada file dan direktori dalam paket penyebaran Anda. Misalnya, untuk memberikan file yang tidak dapat dieksekusi izin yang benar, jalankan perintah berikut.

```
chmod 644 <filepath>
```

Untuk mengubah izin file di Windows, lihat Mengatur, Melihat, Mengubah, atau Menghapus Izin pada Objek dalam dokumentasi Microsoft Windows.

> **Note**
>
> Jika Anda tidak memberikan Lambda izin yang diperlukan untuk mengakses direktori dalam paket penerapan Anda, Lambda menetapkan izin untuk direktori tersebut ke 755 (rwxr-xr-x).

Umum: Terjadi kesalahan saat memanggil UpdateFunctionCode

Kesalahan: Terjadi kesalahan (RequestEntityTooLargeException) saat memanggil UpdateFunctionCode operasi

Ketika Anda mengunggah paket deployment atau arsip lapisan secara langsung ke Lambda, ukuran file ZIP dibatasi ke 50 MB. Untuk mengunggah file yang lebih besar, simpan file di Amazon S3 dan gunakan parameter S3Bucket dan S3Key.

> **Note**
>
> Saat Anda mengunggah file secara langsung dengan AWS CLI, AWS SDK, atau lainnya, file ZIP biner dikonversi ke base64, yang meningkatkan ukurannya sekitar 30%. Untuk memungkinkan hal ini, serta ukuran parameter lain dalam permintaan, batas ukuran permintaan aktual yang digunakan Lambda menjadi lebih besar. Oleh karena itu, batas sebesar 50 MB adalah perkiraan.

Amazon S3: Kode Kesalahan. PermanentRedirect

Kesalahan: Terjadi kesalahan saat GetObject. Kode Kesalahan S3: PermanentRedirect. Pesan Kesalahan S3: Bucket berada di wilayah: us-east-2. Harap gunakan wilayah ini untuk mencoba kembali permintaan ini

Saat Anda mengunggah paket deployment fungsi dari bucket Amazon S3, bucket harus berada di Wilayah yang sama dengan fungsi tersebut. Masalah ini dapat terjadi saat Anda menentukan objek Amazon S3 dalam panggilan ke UpdateFunctionCode, atau menggunakan paket dan menyebarkan perintah di atau AWS CLI CLI. AWS SAM Buat bucket artefak deployment untuk setiap Wilayah tempat Anda mengembangkan aplikasi.

Umum: Tidak dapat menemukan, tidak dapat memuat, tidak dapat mengimpor, kelas tidak ditemukan, tidak ada file atau direktori tersebut

Kesalahan: Tidak dapat menemukan modul 'function'

Kesalahan: tidak dapat memuat file tersebut -- fungsi

Kesalahan: Tidak dapat mengimpor modul 'function'

Kesalahan: Kelas tidak ditemukan: function.Handler

Kesalahan: fork/exec /var/task/function: tidak ada file atau direktori seperti itu

Kesalahan: Tidak dapat memuat tipe 'Function.Handler' dari rakitan 'Fungsi'.

Nama file atau kelas dalam konfigurasi handler fungsi tidak cocok dengan kode. Lihat bagian berikut untuk informasi selengkapnya.

Umum: Metode handler tidak didefinisikan

Kesalahan: index.handler tidak ditentukan atau tidak terekspor

Kesalahan: Handler 'handler' hilang di modul 'function'

Kesalahan: metode `penangan 'yang tidak ditentukan untuk #< > LambdaHandler:0x000055b76ccebf98

Kesalahan: Tidak ada metode publik yang bernama handleRequest dengan tanda tangan metode yang sesuai yang ditemukan di kelas function.Handler

Kesalahan: Tidak dapat menemukan metode 'HandlerEquest' dalam tipe '' dari perakitan 'FungsiFunction.Handler'

Nama metode handler dalam konfigurasi handler fungsi tidak cocok dengan kode. Setiap runtime mendefinisikan konvensi penamaan untuk penangan, seperti. *`filename methodname`*. Handler adalah metode dalam kode fungsi Anda yang dijalankan oleh runtime saat fungsi Anda dipanggil.

Untuk beberapa bahasa, Lambda menyediakan pustaka dengan antarmuka yang mengharapkan metode handler memiliki nama tertentu. Untuk perincian tentang penamaan handler untuk setiap bahasa, lihat topik berikut.

- Membangun Fungsi Acara Lambda dengan Node.js
- Membangun Fungsi Acara Lambda dengan Python
- Membangun Fungsi Acara Lambda dengan Ruby
- Membangun Fungsi Acara Lambda dengan Java
- Membangun Fungsi Acara Lambda dengan Go
- Membangun Fungsi Acara Lambda dengan C #
- Membangun Fungsi Acara Lambda dengan PowerShell

Umum: Batas penyimpanan kode Lambda terlampaui

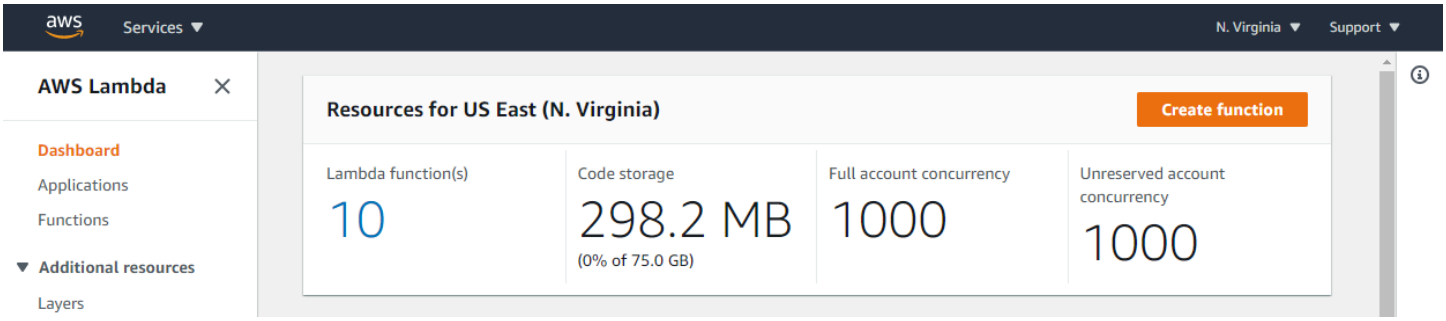
Kesalahan: Batas penyimpanan kode terlampaui.

Lambda menyimpan salinan kode fungsi Anda di bucket S3 internal yang bersifat pribadi ke akun Anda. Setiap AWS akun dialokasikan 300 GB penyimpanan kode (unzip) di setiap Wilayah. Penyimpanan kode mencakup total penyimpanan yang digunakan oleh fungsi dan lapisan Lambda. Jika Anda mencapai kuota, Anda menerima CodeStorageExceededExceptionketika Anda mencoba untuk menerapkan fungsi baru.

Untuk mengatasi masalah ini, Anda dapat membersihkan versi lama fungsi, menghapus kode yang tidak digunakan, atau menggunakan lapisan Lambda untuk menghapus duplikat kode umum yang dibagikan antar fungsi. Selain itu, praktik yang baik untuk menggunakan AWS akun terpisah untuk beban kerja terpisah untuk membantu mengelola kuota penyimpanan.

Atau, Anda dapat mengonfigurasi fungsi dan lapisan Anda untuk menggunakan penyimpanan kode S3 yang dikelola sendiri. Dengan penyimpanan kode S3 yang dikelola sendiri, Lambda mereferensikan kode Anda langsung dari bucket S3 Anda, yang tidak dihitung terhadap kuota penyimpanan Anda.

Anda dapat melihat total penggunaan penyimpanan Anda di konsol Lambda, di bawah submenu Dasbor:



Lambda: Konversi lapisan gagal

Kesalahan: Konversi lapisan Lambda gagal. Untuk saran tentang menyelesaikan masalah ini, lihat halaman Memecahkan masalah penerapan di Lambda di Panduan Pengguna Lambda.

Ketika Anda mengkonfigurasi fungsi Lambda dengan lapisan, Lambda menggabungkan layer dengan kode fungsi Anda. Jika proses ini gagal diselesaikan, Lambda mengembalikan kesalahan ini. Jika Anda mengalami kesalahan ini, lakukan langkah-langkah berikut:

- Hapus file yang tidak digunakan dari lapisan Anda
- Hapus tautan simbolis apa pun di lapisan Anda
- Ganti nama file yang memiliki nama yang sama dengan direktori di salah satu lapisan fungsi Anda

Lambda: atau InvalidParameterValueException RequestEntityTooLargeException

Kesalahan: InvalidParameterValueException: Lambda tidak dapat mengonfigurasi variabel lingkungan Anda karena variabel lingkungan yang Anda berikan melebihi batas 4KB. String diukur: {"A1":" uSFeY5cyPiPn7AtnX5BsM...

Kesalahan: RequestEntityTooLargeException: Permintaan harus lebih kecil dari 5120 byte untuk operasi UpdateFunctionConfiguration

Ukuran maksimum objek variabel yang disimpan dalam konfigurasi fungsi tidak boleh melebihi 4096 byte. Ini termasuk nama kunci, nilai, tanda kutip, koma, dan tanda kurung. Ukuran total badan permintaan HTTP juga terbatas.

```
{
    "FunctionName": "my-function",
    "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-function",
    "Runtime": "nodejs24.x",
    "Role": "arn:aws:iam::123456789012:role/lambda-role",
```

```
    "Environment": {
        "Variables": {
            "BUCKET": "amzn-s3-demo-bucket",
            "KEY": "file.txt"
        }
    },
    ...
}
```

Dalam contoh ini, objek adalah 39 karakter dan menggunakan 39 byte ketika disimpan (tanpa spasi) sebagai string `{"BUCKET":"amzn-s3-demo-bucket","KEY":"file.txt"}`. Karakter ASCII standar dalam nilai variabel lingkungan menggunakan satu byte masing-masing. Karakter ASCII dan Unicode yang diperluas dapat menggunakan antara 2 byte sampai 4 byte per karakter.

Lambda: InvalidParameterValueException

Kesalahan: InvalidParameterValueException: Lambda tidak dapat mengonfigurasi variabel lingkungan Anda karena variabel lingkungan yang Anda berikan berisi kunci cadangan yang saat ini tidak didukung untuk modifikasi.

Lambda mencadangkan beberapa kunci variabel lingkungan untuk penggunaan internal. Misalnya, `AWS_REGION` digunakan oleh runtime untuk menentukan Wilayah saat ini dan tidak dapat dibatalkan. Variabel lainnya, seperti `PATH`, digunakan oleh runtime, tetapi dapat diperluas dalam konfigurasi fungsi Anda. Untuk daftar lengkap, lihat Variabel lingkungan runtime yang ditetapkan.

Lambda: Konkurensi dan kuota memori

Kesalahan: Ditentukan ConcurrentExecutions untuk fungsi mengurangi akun UnreservedConcurrentExecution di bawah nilai minimumnya

Kesalahan: Nilai MemorySize '' gagal memenuhi kendala: Anggota harus memiliki nilai kurang dari atau sama dengan 3008

Kesalahan ini terjadi ketika Anda melebihi kuota konkurensi atau memori untuk akun Anda. AWS Akun baru telah mengurangi kuota konkurensi dan memori. Untuk mengatasi kesalahan yang terkait dengan konkurensi, Anda dapat meminta peningkatan kuota. Anda tidak dapat meminta peningkatan kuota memori.

- Konkurensi: Anda mungkin mendapatkan kesalahan jika mencoba membuat fungsi menggunakan konkurensi cadangan atau ketentuan, atau jika permintaan konkurensi per fungsi (PutFunctionConcurrency) melebihi kuota konkurensi akun Anda.

- Memori: Kesalahan terjadi jika jumlah memori yang dialokasikan ke fungsi melebihi kuota memori akun Anda.

Lambda: Konfigurasi alias tidak valid untuk konkurensi yang disediakan

Kesalahan: Konfigurasi alias tidak valid untuk konkurensi yang disediakan

Kesalahan ini terjadi ketika Anda mencoba memperbarui kode atau konfigurasi fungsi Lambda sementara alias dengan konkurensi yang disediakan menunjuk ke versi yang memiliki masalah. Lambda melakukan pra-inisialisasi lingkungan eksekusi untuk konkurensi yang disediakan, dan jika lingkungan ini tidak dapat diinisialisasi dengan benar karena kesalahan kode, kendala sumber daya, atau tumpukan dan alias yang terpengaruh, penerapan gagal. Jika Anda mengalami masalah ini, lakukan langkah-langkah berikut:

1. Kembalikan alias: Perbarui sementara alias untuk menunjuk ke versi yang sebelumnya berfungsi.

   ```
   aws lambda update-alias \
    --function-name <function-name> \
    --name <alias-name> \
    --function-version <known-good-version>
   ```

2. Perbaiki kode inisialisasi Lambda: Pastikan kode inisialisasi yang berjalan di luar handler tidak memiliki pengecualian yang tidak tertangkap dan menginisialisasi klien dan koneksi.
3. Menerapkan ulang keamanan: Menyebarkan kode tetap dan menerbitkan versi baru. Kemudian, perbarui alias untuk menunjuk ke versi tetap. Secara opsional, aktifkan kembali konkurensi yang disediakan, jika perlu.

Jika menggunakan AWS CloudFormation, perbarui definisi tumpukan `FunctionVersion:!GetAtt version.Version` sehingga alias menunjuk ke versi yang berfungsi:

```
alias:
 Type: AWS::Lambda::Alias
 Properties:
 FunctionName: !Ref function
FunctionVersion: !GetAtt version.Version
 Name: BLUE
 ProvisionedConcurrencyConfig:
 ProvisionedConcurrentExecutions: 1
```

Menyelesaikan masalah invokasi di Lambda

Saat Anda memanggil Fungsi Peristiwa Lambda, Lambda memvalidasi permintaan dan memeriksa kapasitas penskalaan sebelum mengirim acara ke fungsi Anda atau, untuk pemanggilan asinkron, ke antrian acara. Kesalahan invokasi dapat disebabkan oleh masalah parameter permintaan, struktur kejadian, pengaturan fungsi, izin pengguna, izin sumber daya, atau batas.

Jika Anda memanggil fungsi secara langsung, Anda melihat kesalahan invokasi apa pun dalam respons dari Lambda. Jika Anda memanggil fungsi secara asinkron dengan pemetaan sumber kejadian atau melalui layanan lain, Anda mungkin menemukan kesalahan di log, antrean surat gagal, atau tujuan kejadian-gagal. Opsi penanganan kesalahan dan perilaku percobaan ulang berbeda-beda tergantung pada cara Anda memanggil fungsi dan jenis kesalahan.

Untuk daftar tipe kesalahan yang dapat dikembalikan oleh operasi `Invoke`, lihat Panggil.

Topik

- Lambda: Waktu fungsi habis selama fase Init () Sandbox.Timedout
- IAM: lambda: InvokeFunction tidak diizinkan
- Lambda: Tidak dapat menemukan bootstrap () yang valid Runtime.InvalidEntrypoint
- Lambda: Operasi tidak dapat dilakukan ResourceConflictException
- Lambda: Fungsi tertahan dalam Tertunda
- Lambda: Salah satu fungsi menggunakan semua konkurensi
- Umum: Tidak dapat memanggil fungsi dengan akun atau layanan lain
- Umum: Invokasi fungsi adalah perulangan
- Lambda: Perutean alias dengan konkurensi terprovisi
- Lambda: Mulai awal dengan konkurensi terprovisi
- Lambda: Mulai awal dengan versi baru
- Lambda: Node.js Keluar tak terduga di runtime () Runtime.NodejsExit
- EFS: Fungsi tidak dapat memasang sistem file EFS
- EFS: Fungsi tidak dapat terhubung ke sistem file EFS
- EFS: Fungsi tidak dapat memasang sistem file EFS karena waktu habis.
- File S3: Fungsi tidak dapat me-mount sistem file S3
- File S3: Fungsi tidak dapat terhubung ke sistem file S3
- File S3: Fungsi tidak dapat me-mount sistem file S3 karena batas waktu

- Lambda: Lambda mendeteksi proses IO yang memakan waktu terlalu lama
- Wadah: CodeArtifactUserException kesalahan
- Wadah: InvalidEntrypoint kesalahan

Lambda: Waktu fungsi habis selama fase Init () Sandbox.Timedout

Kesalahan: Tugas habis setelah 3,00 detik

Saat fase Init habis, Lambda menginisialisasi lingkungan eksekusi lagi dengan menjalankan kembali `Init` fase saat permintaan pemanggilan berikutnya tiba. Ini disebut init yang ditekan. Namun, jika fungsi Anda dikonfigurasi dengan durasi waktu tunggu yang singkat (umumnya sekitar 3 detik), init yang ditekan mungkin tidak selesai selama durasi batas waktu yang dialokasikan, menyebabkan `Init` fase habis lagi. Atau, init yang ditekan selesai tetapi tidak menyisakan cukup waktu untuk menyelesaikan fase Invoke, menyebabkan `Invoke` fase habis.

Untuk mengurangi kesalahan batas waktu, gunakan satu atau beberapa strategi berikut:

- Tingkatkan durasi batas waktu fungsi: Perpanjang batas waktu untuk memberikan waktu `Init` dan `Invoke` fase agar berhasil diselesaikan.
- Tingkatkan alokasi memori fungsi: Lebih banyak memori juga berarti alokasi CPU yang lebih proporsional, yang dapat mempercepat fase dan fase. `Init` `Invoke`
- Optimalkan kode inisialisasi fungsi: Kurangi waktu yang diperlukan untuk inisialisasi untuk memastikan bahwa `Invoke` fase `Init` dan dapat selesai dalam batas waktu yang dikonfigurasi.

IAM: lambda: InvokeFunction tidak diizinkan

Kesalahan: Pengguna: arn:aws:iam: :123456789012: tidak berwenang untuk melakukan: lambda: on resource: user/developer my-function InvokeFunction

Pengguna Anda, atau peran yang Anda asumsikan, harus memiliki izin untuk menjalankan fungsi. Persyaratan ini juga berlaku untuk fungsi Lambda dan sumber daya komputasi lainnya yang memanggil fungsi. Tambahkan kebijakan AWS terkelola AWSLambdaRoleke pengguna Anda, atau tambahkan kebijakan khusus yang memungkinkan `lambda:InvokeFunction` tindakan pada fungsi target.

> **Note**
>
> Nama tindakan IAM (`lambda:InvokeFunction`) mengacu pada operasi API `Invoke` Lambda.

Untuk informasi selengkapnya, lihat Mengelola izin di AWS Lambda.

Lambda: Tidak dapat menemukan bootstrap () yang valid Runtime.InvalidEntrypoint

Kesalahan: Tidak dapat menemukan bootstrap yang valid: [/var/task/bootstrap/opt/bootstrap]

Kesalahan ini biasanya terjadi ketika root paket penyebaran Anda tidak berisi file yang dapat dieksekusi bernama. `bootstrap` Misalnya, jika Anda menerapkan `provided.al2023` fungsi dengan file.zip, `bootstrap` file harus berada di root file.zip, bukan di direktori.

Lambda: Operasi tidak dapat dilakukan ResourceConflictException

KesalahanResourceConflictException:: Operasi tidak dapat dilakukan saat ini. Fungsi saat sedang dalam status: Tertunda

Saat Anda menghubungkan fungsi ke virtual private cloud (VPC) pada saat pembuatan, fungsi memasukkan status `Pending` sewaktu Lambda membuat antarmuka jaringan elastis. Selama waktu ini, Anda tidak dapat memanggil atau memodifikasi fungsi Anda. Jika Anda menghubungkan fungsi Anda ke VPC setelah pembuatan, Anda dapat memanggilnya saat pembaruan tertunda, tetapi Anda tidak dapat memodifikasi kode atau konfigurasinya.

Untuk informasi selengkapnya, lihat Status Fungsi Acara Lambda.

Lambda: Fungsi tertahan dalam Tertunda

Kesalahan: Fungsi tertahan dalam status *`Pending`* selama beberapa menit.

Jika suatu fungsi tertahan di status `Pending` selama lebih dari enam menit, panggil salah satu operasi API berikut untuk membukanya:

- UpdateFunctionCode
- UpdateFunctionConfiguration
- PublishVersion

Lambda membatalkan operasi yang tertunda dan menempatkan fungsi ke keadaan `Failed`. Anda kemudian dapat mencoba pembaruan lain.

Lambda: Salah satu fungsi menggunakan semua konkurensi

Masalah: Satu fungsi sedang menggunakan semua konkurensi yang tersedia, yang menyebabkan fungsi lain dibatasi.

Untuk membagi konkurensi AWS akun Anda yang tersedia di AWS Wilayah menjadi kumpulan, gunakan konkurensi cadangan. Konkurensi cadangan memastikan fungsi dapat selalu menskalakan ke konkurensi yang ditetapkan, dan fungsi tidak menskalakan melebihi konkurensi yang ditetapkan.

Umum: Tidak dapat memanggil fungsi dengan akun atau layanan lain

Masalah: Anda dapat memanggil fungsi secara langsung, tetapi tidak dijalankan ketika layanan atau akun lain memanggilnya.

Anda memberikan izin kepada akun dan layanan lain untuk memanggil fungsi dalam kebijakan berbasis sumber daya milik fungsi. Jika pemanggil ada di akun lain, pengguna tersebut juga harus memiliki izin untuk memanggil fungsi.

Umum: Invokasi fungsi adalah perulangan

Masalah: Fungsi dipanggil secara berulang terus-menerus.

Ini biasanya terjadi ketika fungsi Anda mengelola sumber daya dalam AWS layanan yang sama yang memicunya. Misalnya, ini memungkinkan untuk membuat fungsi yang menyimpan objek dalam bucket Amazon Simple Storage Service (Amazon S3) yang dikonfigurasi dengan notifikasi yang kembali memanggil fungsi. Untuk menghentikan fungsi agar tidak berjalan, kurangi konkurensi yang tersedia menjadi nol, yang membatasi semua pemanggilan future.

Selanjutnya, identifikasi jalur kode atau kesalahan konfigurasi yang menyebabkan invokasi berulang. Lambda secara otomatis mendeteksi dan menghentikan loop rekursif untuk beberapa AWS layanan dan SDK. Untuk informasi selengkapnya, lihat the section called “Deteksi loop rekursif”.

Lambda: Perutean alias dengan konkurensi terprovisi

Masalah: Konkurensi terprovisi melimpahkan invokasi selama perutean alias.

Lambda menggunakan model probabilistik sederhana untuk mendistribusikan lalu lintas di antara dua versi fungsi. Pada tingkat lalu lintas rendah, Anda mungkin melihat varians tinggi di antara persentase lalu lintas yang dikonfigurasi dan aktual di setiap versi. Jika fungsi Anda menggunakan konkurensi

terprovisi, Anda dapat menghindari invokasi limpahan dengan mengonfigurasi jumlah yang lebih tinggi dari instans konkurensi terprovisi selama perutean alias aktif.

Lambda: Mulai awal dengan konkurensi terprovisi

Masalah: Anda melihat mulai awal setelah mengaktifkan konkurensi terprovisi.

Ketika jumlah eksekusi bersamaan pada fungsi kurang dari atau sama dengan tingkat konkurensi terprovisi yang dikonfigurasi, mulai awal seharusnya tidak ada. Untuk membantu Anda mengonfirmasi jika konkurensi terprovisi beroperasi secara normal, lakukan hal berikut:

- Periksa apakah concurrency terprovisi diaktifkan pada versi fungsi atau alias.

  > **Note**
  >
  > Konkurensi yang disediakan tidak dapat dikonfigurasi pada versi fungsi yang tidak dipublikasikan ($LATEST).

- Pastikan pemicu Anda memanggil versi fungsi atau alias yang benar. Misalnya, jika Anda menggunakan Amazon API Gateway, periksa API Gateway memanggil versi fungsi atau alias dengan konkurensi terprovisi, bukan $LATEST. Untuk mengonfirmasi bahwa konkurensi yang disediakan sedang digunakan, Anda dapat memeriksa metrik AmazonProvisionedConcurrencyInvocations . CloudWatch Nilai bukan nol menunjukkan fungsi ini memproses invokasi pada lingkungan eksekusi yang diinisialisasi.
- Tentukan apakah konkurensi fungsi Anda melebihi tingkat konkurensi yang disediakan yang dikonfigurasi dengan memeriksa metrik. ProvisionedConcurrencySpilloverInvocations CloudWatch Nilai bukan nol menunjukkan semua konkurensi terprovisi sedang digunakan dan beberapa invokasi terjadi dengan mulai awal.
- Periksa frekuensi invokasi (permintaan per detik). Fungsi dengan konkurensi terprovisi memiliki tingkat maksimum 10 permintaan per detik per konkurensi terprovisi. Misalnya, fungsi yang dikonfigurasi dengan 100 konkurensi terprovisi dapat menangani 1.000 permintaan per detik. Jika tingkat invokasi melebihi 1.000 permintaan per detik, beberapa mulai awal dapat terjadi.

Lambda: Mulai awal dengan versi baru

Masalah: Anda melihat mulai awal saat men-deploy versi baru dari fungsi Anda.

Ketika Anda memperbarui alias fungsi, Lambda secara otomatis menggeser konkurensi terprovisi ke versi baru berdasarkan bobot yang dikonfigurasi di alias.

Kesalahan:KMSDisabledException: Lambda tidak dapat mendekripsi variabel lingkungan karena kunci KMS yang digunakan dinonaktifkan. Silakan periksa pengaturan kunci KMS fungsi.

Kesalahan ini dapat terjadi jika kunci AWS Key Management Service (AWS KMS) Anda dinonaktifkan, atau jika hibah yang memungkinkan Lambda menggunakan kunci dicabut. Jika pemberian izin tidak ada, konfigurasikan fungsi agar menggunakan kunci yang berbeda. Selanjutnya, atur ulang kunci kustom untuk membuat ulang pemberian izin.

Lambda: Node.js Keluar tak terduga di runtime () Runtime.NodejsExit

Masalah: Klien runtime Lambda mendeteksi kode keluar yang tidak terduga Node.js .

Kesalahan ini terjadi ketika fungsi Anda keluar sebelum semua Promises diselesaikan, misalnya karena bug kode. Ini juga dapat terjadi ketika Node.js mendeteksi kebuntuan yang mencegah Janji diselesaikan. Kesalahan ini hanya memengaruhi penangan gaya asinkron, bukan penangan gaya panggilan balik.

Runtime yang terpengaruh: Node.js 18 dan yang lebih baru.

Untuk mengatasi masalah ini:

1. Periksa kode fungsi Anda untuk janji yang tidak terselesaikan di penangan asinkron.
2. Pastikan semua janji diselesaikan dengan benar (diselesaikan atau ditolak) sebelum fungsi selesai.
3. Tinjau kode Anda untuk kondisi balapan potensial dalam operasi asinkron.

Untuk informasi selengkapnya tentang kode Node.js keluar dan penghentian proses, lihat Node.js dokumentasi.

EFS: Fungsi tidak dapat memasang sistem file EFS

Kesalahan:EFSMountFailureException: Fungsi tidak dapat me-mount sistem file EFS dengan titik akses arn:aws:elasticfilesystem:us-east- 2:123456789012:access- -015cxmplb72b405fd. point/fsap

Permintaan pemasangan untuk sistem file fungsi ditolak. Periksa izin fungsi, dan konfirmasikan bahwa sistem file dan titik aksesnya sudah ada dan siap digunakan.

EFS: Fungsi tidak dapat terhubung ke sistem file EFS

Kesalahan:EFSMountConnectivityException: Fungsi tidak dapat terhubung ke sistem file Amazon EFS dengan titik akses arn:aws:elasticfilesystem:us-east- 2:123456789012:access- -015cxmplb72b405fd. point/fsap Periksa konfigurasi jaringan Anda dan coba kembali.

Fungsi tidak dapat membuat koneksi ke sistem file fungsi dengan protokol NFS (TCP port 2049). Periksa grup keamanan dan konfigurasi perutean untuk subnet VPC.

Jika Anda mendapatkan kesalahan ini setelah memperbarui pengaturan konfigurasi VPC fungsi Anda, coba lepaskan dan pasang kembali sistem file.

EFS: Fungsi tidak dapat memasang sistem file EFS karena waktu habis.

Kesalahan:EFSMountTimeoutException: Fungsi tidak dapat me-mount sistem file EFS dengan titik akses {arn:aws:elasticfilesystem:us-east- 2:123456789012:access- -015cxmplb72b405fd} karena waktu pemasangan habis. point/fsap

Fungsi dapat terhubung ke sistem file fungsi, tetapi waktu operasi pemasangan habis. Coba lagi setelah beberapa saat dan pertimbangkan untuk membatasi konkurensi fungsi guna mengurangi beban sistem file.

File S3: Fungsi tidak dapat me-mount sistem file S3

Kesalahan:S3FilesMountFailureException: Fungsi tidak dapat me-mount sistem file Amazon S3 dengan titik akses arn:aws:s3files:us-east- 2:123456789012: access- -123456789abcde. point/fsap

Permintaan pemasangan untuk sistem file fungsi ditolak. Periksa izin fungsi, dan konfirmasikan bahwa sistem file dan titik aksesnya sudah ada dan siap digunakan.

File S3: Fungsi tidak dapat terhubung ke sistem file S3

Kesalahan:S3FilesMountConnectivityException: Fungsi tidak dapat terhubung ke sistem file Amazon S3 dengan titik akses arn:aws:s3files:us-east- 2:123456789012: access- -123456789abcde. point/fsap Periksa konfigurasi jaringan Anda dan coba kembali.

Fungsi tidak dapat membuat koneksi ke sistem file fungsi dengan protokol NFS (TCP port 2049). Periksa grup keamanan dan konfigurasi perutean untuk subnet VPC.

Jika Anda mendapatkan kesalahan ini setelah memperbarui pengaturan konfigurasi VPC fungsi Anda, coba lepaskan dan pasang kembali sistem file.

File S3: Fungsi tidak dapat me-mount sistem file S3 karena batas waktu

Kesalahan:S3FilesMountTimeoutException: Fungsi tidak dapat me-mount sistem file S3 dengan titik akses {arn:aws:s3files:us-east- 2:123456789012:access- -123456789abcde} karena waktu pemasangan habis. point/fsap

Fungsi dapat terhubung ke sistem file fungsi, tetapi waktu operasi pemasangan habis. Coba lagi setelah beberapa saat dan pertimbangkan untuk membatasi konkurensi fungsi guna mengurangi beban sistem file.

Lambda: Lambda mendeteksi proses IO yang memakan waktu terlalu lama

EFSIOException: Instance fungsi ini dihentikan karena Lambda mendeteksi proses IO yang memakan waktu terlalu lama.

Waktu invokasi sebelumnya habis dan Lambda tidak dapat menghentikan handler fungsi. Masalah ini dapat terjadi ketika sistem file yang dilampirkan kehabisan kredit lonjakan dan throughput awal tidak cukup. Untuk meningkatkan throughput, Anda dapat meningkatkan ukuran sistem file atau menggunakan throughput terprovisi.

Wadah: CodeArtifactUserException kesalahan

Kesalahan: pesan CodeArtifactUserPendingException kesalahan

CodeArtifact Optimasi tertunda. Fungsi transisi ke status Aktif saat Lambda menyelesaikan pengoptimalan. Kode respons HTTP 409.

Kesalahan: pesan CodeArtifactUserDeletedException kesalahan

CodeArtifact Itu dijadwalkan akan dihapus. Kode respons HTTP 409.

Kesalahan: pesan CodeArtifactUserFailedException kesalahan

Lambda gagal mengoptimalkan kode. Anda perlu memperbaiki kode dan mengunggahnya lagi. Kode respons HTTP 409.

Wadah: InvalidEntrypoint kesalahan

Kesalahan: Runtime.ExitError atau “ErrorType”: "" Runtime.InvalidEntrypoint

Verifikasi bahwa ENTRYPOINT untuk gambar kontainer Anda mencakup jalur absolut sebagai lokasi. Selain itu, verifikasi bahwa gambar tidak berisi symlink sebagai ENTRYPOINT.

Kesalahan: Anda menggunakan CloudFormation template, dan wadah ENTRYPOINT Anda sedang diganti dengan nilai nol atau kosong.

Tinjau ImageConfigsumber daya dalam CloudFormation template. Jika Anda menyatakan sumber daya `ImageConfig` di templat Anda, Anda harus menyediakan nilai non-kosong untuk ketiga properti.

Menyelesaikan masalah eksekusi di Lambda

Saat runtime Lambda menjalankan kode fungsi Anda, acara tersebut mungkin diproses pada instance Fungsi Peristiwa Lambda yang telah memproses peristiwa selama beberapa waktu, atau mungkin memerlukan instance baru untuk diinisialisasi. Kesalahan dapat terjadi selama inisialisasi fungsi, ketika kode handler Anda memproses kejadian, atau ketika fungsi Anda mengembalikan (atau gagal mengembalikan) respons.

Kesalahan eksekusi fungsi dapat disebabkan oleh masalah dengan kode, konfigurasi fungsi, sumber daya hilir, atau izin Anda. Jika Anda memanggil fungsi secara langsung, Anda melihat kesalahan fungsi dalam respons dari Lambda. Jika Anda memanggil fungsi Anda secara asinkron, dengan pemetaan sumber kejadian, atau melalui layanan lain, Anda mungkin menemukan kesalahan di log, antrean surat gagal, atau tujuan saat terjadi kegagalan. Opsi penanganan kesalahan dan perilaku percobaan ulang berbeda-beda tergantung pada cara Anda memanggil fungsi dan jenis kesalahan.

Saat kode fungsi atau runtime Lambda Anda memunculkan kesalahan, kode status di respons dari Lambda adalah 200 OK. Adanya kesalahan dalam respons ditunjukkan dengan header bernama `X-Amz-Function-Error`. Kode status seri 400 dan 500 dicadangkan untuk kesalahan invokasi.

Topik

- Lambda: Debugging jarak jauh dengan Kode Visual Studio
- Lambda: Eksekusi memerlukan waktu yang lama
- Lambda: Payload acara tak terduga
- Lambda: Ukuran muatan besar yang tak terduga
- Lambda: Kesalahan pengkodean dan decoding JSON
- Lambda: Log atau jejak tidak muncul
- Lambda: Tidak semua log fungsi saya muncul
- Lambda: Fungsi kembali sebelum eksekusi selesai
- Lambda: Menjalankan versi atau alias fungsi yang tidak diinginkan

- Lambda: Mendeteksi loop tak terbatas
- Umum: Ketidaktersediaan layanan hilir
- AWS SDK: Versi dan pembaruan
- Python: Pustaka memuat dengan tidak benar
- Java: Fungsi Anda membutuhkan waktu lebih lama untuk memproses peristiwa setelah memperbarui ke Java 17 dari Java 11
- Kafka: Penanganan kesalahan dan coba lagi masalah konfigurasi

Lambda: Debugging jarak jauh dengan Kode Visual Studio

Masalah: Kesulitan memecahkan masalah perilaku fungsi Lambda yang kompleks di lingkungan aktual AWS

Lambda menyediakan fitur debugging jarak jauh melalui file. AWS Toolkit for Visual Studio Code Untuk pengaturan dan instruksi umum, lihatDebug dari jarak jauh Fungsi Acara Lambda dengan Kode Visual Studio.

Untuk petunjuk mendetail tentang pemecahan masalah, kasus penggunaan lanjutan, dan ketersediaan wilayah, lihat Fungsi Lambda debugging jarak jauh di Panduan Pengguna. AWS Toolkit for Visual Studio Code

Lambda: Eksekusi memerlukan waktu yang lama

Masalah: Eksekusi fungsi membutuhkan waktu terlalu lama.

Jika kode Anda membutuhkan waktu lebih lama untuk berjalan di Lambda daripada di mesin lokal Anda, mungkin dibatasi oleh memori atau daya pemrosesan yang tersedia untuk fungsi tersebut. Konfigurasikan fungsi dengan memori tambahan untuk meningkatkan memori dan CPU.

Lambda: Payload acara tak terduga

Masalah: Kesalahan fungsi terkait dengan JSON yang salah bentuk atau validasi data yang tidak memadai.

Semua fungsi Lambda menerima payload peristiwa di parameter pertama handler. Payload acara adalah struktur JSON yang mungkin berisi array dan elemen bersarang.

JSON yang salah dapat terjadi ketika disediakan oleh layanan hulu yang tidak menggunakan proses yang kuat untuk memeriksa struktur JSON. Ini terjadi ketika layanan menggabungkan string teks

atau menyematkan input pengguna yang belum dibersihkan. JSON juga sering diserialisasikan untuk lewat antar layanan. Selalu mengurai struktur JSON baik sebagai produsen maupun konsumen JSON untuk memastikan bahwa strukturnya valid.

Demikian pula, gagal memeriksa rentang nilai dalam muatan acara dapat mengakibatkan kesalahan. Contoh ini menunjukkan fungsi yang menghitung pemotongan pajak:

```
exports.handler = async (event) => {
    let pct = event.taxPct
    let salary = event.salary

    // Calculate % of paycheck for taxes
    return (salary * pct)
}
```

Fungsi ini menggunakan gaji dan tarif pajak dari muatan acara untuk melakukan perhitungan. Namun, kode gagal untuk memeriksa apakah atribut ada. Ini juga gagal untuk memeriksa tipe data, atau memastikan batasan, seperti memastikan bahwa persentase pajak antara 0 dan 1. Akibatnya, nilai di luar batas-batas ini menghasilkan hasil yang tidak masuk akal. Jenis yang salah atau atribut yang hilang menyebabkan kesalahan runtime.

Buat pengujian untuk memastikan bahwa fungsi Anda menangani ukuran muatan yang lebih besar. Ukuran maksimum untuk muatan acara Lambda adalah 1 MB. Bergantung pada kontennya, muatan yang lebih besar mungkin berarti lebih banyak item yang diteruskan ke fungsi atau lebih banyak data biner yang disematkan dalam atribut JSON. Dalam kedua kasus, ini dapat menghasilkan lebih banyak pemrosesan untuk fungsi Lambda.

Muatan yang lebih besar juga dapat menyebabkan batas waktu. Misalnya, fungsi Lambda memproses satu catatan per 100 ms dan memiliki batas waktu 3 detik. Pemrosesan berhasil untuk 0-29 item dalam muatan. Namun, setelah payload berisi lebih dari 30 item, fungsi akan habis dan menimbulkan kesalahan. Untuk menghindari hal ini, pastikan batas waktu diatur untuk menangani waktu pemrosesan tambahan untuk jumlah maksimum item yang diharapkan.

Lambda: Ukuran muatan besar yang tak terduga

Masalah: Fungsi habis waktu atau menyebabkan kesalahan karena muatan besar.

Muatan yang lebih besar dapat menyebabkan batas waktu dan kesalahan. Sebaiknya buat pengujian untuk memastikan bahwa fungsi Anda menangani muatan terbesar yang diharapkan, dan memastikan batas waktu fungsi disetel dengan benar.

Selain itu, muatan peristiwa tertentu dapat berisi petunjuk ke sumber daya lain. Misalnya, fungsi Lambda dengan memori 128 MB mungkin melakukan pemrosesan gambar pada file JPG yang disimpan sebagai objek di S3. Fungsi ini berfungsi seperti yang diharapkan dengan file gambar yang lebih kecil.

Namun, ketika file JPG yang lebih besar disediakan sebagai input, fungsi Lambda memunculkan kesalahan karena kehabisan memori. Untuk menghindari hal ini, kasus uji harus menyertakan contoh dari batas atas ukuran data yang diharapkan. Kode juga harus memvalidasi ukuran payload.

Lambda: Kesalahan pengkodean dan decoding JSON

Masalah: NoSuchKey pengecualian saat mengurai input JSON.

Periksa untuk memastikan Anda memproses atribut JSON dengan benar. Misalnya, untuk peristiwa yang dihasilkan oleh S3, `s3.object.key` atribut berisi nama kunci objek yang dikodekan URL. Banyak fungsi memproses atribut ini sebagai teks untuk memuat objek S3 yang direferensikan:

Example

```
const originalText = await s3.getObject({
  Bucket: event.Records[0].s3.bucket.name,
  Key: event.Records[0].s3.object.key
}).promise()
```

Kode ini bekerja dengan nama kunci `james.jpg` tetapi menimbulkan `NoSuchKey` kesalahan saat namanya. `james beswick.jpg` Karena pengkodean URL mengonversi spasi dan karakter lain dalam nama kunci, Anda harus memastikan bahwa fungsi memecahkan kode kunci sebelum menggunakan data ini:

Example

```
const originalText = await s3.getObject({
  Bucket: event.Records[0].s3.bucket.name,
  Key: decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/g, " "))
}).promise()
```

Lambda: Log atau jejak tidak muncul

Masalah: Log tidak muncul di CloudWatch Log.

Masalah: Jejak tidak muncul di AWS X-Ray.

Fungsi Anda memerlukan izin untuk memanggil CloudWatch Log dan X-Ray. Perbarui peran eksekusi untuk memberikan izin. Tambahkan kebijakan terkelola berikut untuk mengaktifkan log dan pelacakan.

- AWSLambdaBasicExecutionRole
- AWSXRayDaemonWriteAccess

Saat Anda menambahkan izin ke fungsi Anda, lakukan pembaruan sepele ke kode atau konfigurasinya juga. Ini memaksa instance menjalankan fungsi Anda, yang memiliki kredensialnya yang sudah ketinggalan zaman, untuk berhenti dan diganti.

> **Note**
>
> Mungkin perlu 5 hingga 10 menit agar log muncul setelah pemanggilan fungsi.

Lambda: Tidak semua log fungsi saya muncul

Masalah: Log fungsi tidak ada di CloudWatch Log, meskipun izin saya benar

Jika Anda Akun AWS mencapai batas kuota CloudWatch Log, fungsi CloudWatch throttles logging berfungsi. Ketika ini terjadi, beberapa log yang dihasilkan oleh fungsi Anda mungkin tidak muncul di CloudWatch Log.

Jika fungsi Anda mengeluarkan log pada tingkat yang terlalu tinggi bagi Lambda untuk memprosesnya, ini juga dapat menyebabkan keluaran log tidak muncul di Log. CloudWatch Ketika Lambda tidak dapat mengirim log CloudWatch pada tingkat yang dihasilkan oleh fungsi Anda, Lambda akan menjatuhkan log untuk mencegah eksekusi fungsi Anda melambat. Berharap untuk secara konsisten mengamati log yang dijatuhkan ketika throughput log Anda melebihi 2 MB/s untuk satu aliran log.

Jika fungsi Anda dikonfigurasi untuk menggunakan log berformat JSON, Lambda mencoba mengirim `logsDropped`peristiwa ke CloudWatch Log saat menjatuhkan log. Namun, saat CloudWatch membatasi logging fungsi Anda, peristiwa ini mungkin tidak mencapai CloudWatch Log, sehingga Anda tidak akan selalu melihat catatan saat Lambda menjatuhkan log.

Untuk memeriksa apakah Anda Akun AWS telah mencapai batas kuota CloudWatch Log, lakukan hal berikut:

1. Buka Konsol Service Quotas.
2. Di panel navigasi, pilih Layanan AWS .
3. Dari daftar AWS layanan, cari Amazon CloudWatch Logs.
4. Dalam daftar kuota Layanan, pilih`CreateLogGroup throttle limit in transactions per second`, `CreateLogStream throttle limit in transactions per second` dan `PutLogEvents throttle limit in transactions per second` kuota untuk melihat penggunaan Anda.

Anda juga dapat mengatur CloudWatch alarm untuk mengingatkan Anda ketika penggunaan akun Anda melebihi batas yang Anda tentukan untuk kuota ini. Lihat Membuat CloudWatch alarm berdasarkan ambang batas statis untuk mempelajari lebih lanjut.

Jika batas kuota default untuk CloudWatch Log tidak cukup untuk kasus penggunaan, Anda dapat meminta peningkatan kuota.

Lambda: Fungsi kembali sebelum eksekusi selesai

Masalah: (Node.js) Fungsi kembali sebelum kode selesai dieksekusi

Banyak pustaka, termasuk AWS SDK, beroperasi secara asinkron. Ketika Anda melakukan panggilan jaringan atau melakukan operasi lain yang perlu menunggu respons, pustaka mengembalikan objek yang disebut janji, yang melacak kemajuan operasi di latar belakang.

Untuk menunggu janji selesai menjadi respons, gunakan kata kunci `await`. Ini menghalangi kode handler Anda beroperasi hingga janji diselesaikan menjadi objek yang berisi respons. Jika Anda tidak perlu menggunakan data dari respons dalam kode, Anda dapat mengembalikan janji secara langsung ke runtime.

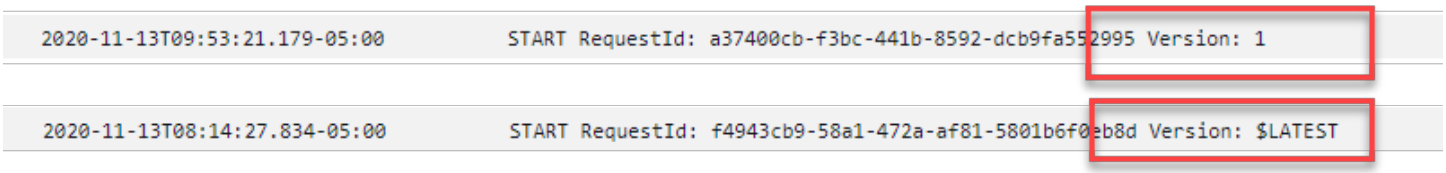
Beberapa pustaka tidak mengembalikan janji, tetapi dapat dirangkum dalam kode yang mengembalikan janji. Untuk informasi selengkapnya, lihat Tentukan penangan Fungsi Peristiwa Lambda di Node.js.

Lambda: Menjalankan versi atau alias fungsi yang tidak diinginkan

Masalah: Versi fungsi atau alias tidak dipanggil

Saat Anda mempublikasikan fungsi Lambda baru di konsol atau menggunakan AWS SAM, versi kode terbaru diwakili oleh. `$LATEST` Secara default, pemanggilan yang tidak menentukan versi atau alias secara otomatis menargetkan `$LATEST` versi kode fungsi Anda.

Jika Anda menggunakan versi fungsi atau alias tertentu, ini adalah versi fungsi yang diterbitkan yang tidak dapat diubah sebagai tambahan. $LATEST Saat memecahkan masalah fungsi ini, pertama-tama tentukan bahwa pemanggil telah memanggil versi atau alias yang dimaksud. Anda dapat melakukan ini dengan memeriksa log fungsi Anda. Versi fungsi yang dipanggil selalu ditampilkan di baris log START:

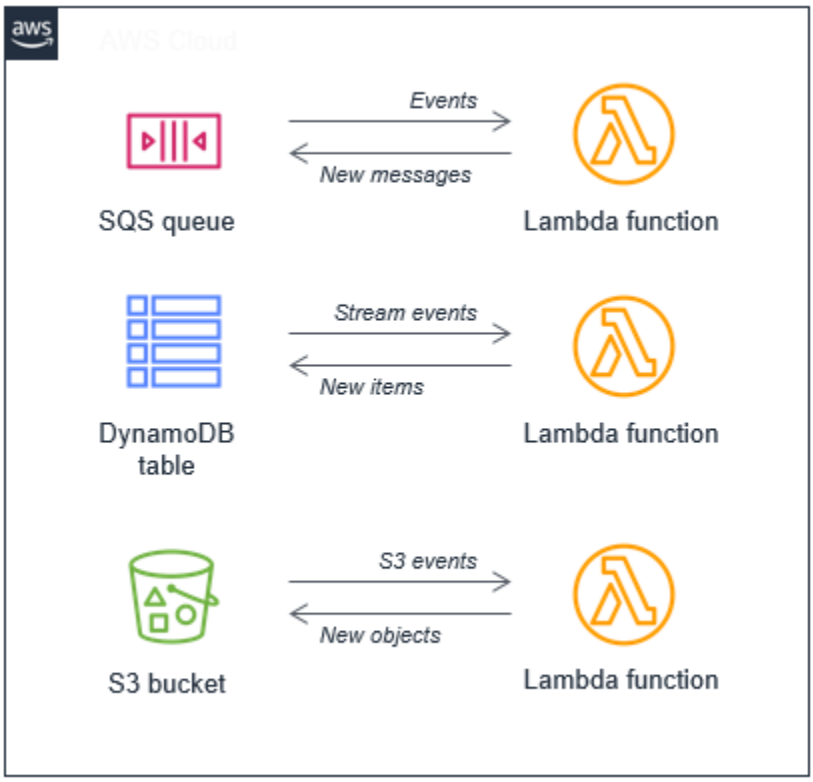


Lambda: Mendeteksi loop tak terbatas

Masalah: Pola loop tak terbatas terkait dengan fungsi Lambda

Ada dua jenis loop tak terbatas dalam fungsi Lambda. Yang pertama ada di dalam fungsi itu sendiri, disebabkan oleh loop yang tidak pernah keluar. Pemanggilan berakhir hanya ketika fungsi habis waktu. Anda dapat mengidentifikasi ini dengan memantau batas waktu, dan kemudian memperbaiki perilaku perulangan.

Jenis loop kedua adalah antara fungsi Lambda dan sumber daya lainnya AWS . Ini terjadi ketika peristiwa dari sumber daya seperti bucket S3 memanggil fungsi Lambda, yang kemudian berinteraksi dengan sumber daya sumber yang sama untuk memicu peristiwa lain. Ini memanggil fungsi lagi, yang menciptakan interaksi lain dengan bucket S3 yang sama, dan seterusnya. Jenis loop ini dapat disebabkan oleh sejumlah sumber AWS peristiwa yang berbeda, termasuk antrian Amazon SQS dan tabel DynamoDB. Anda dapat menggunakan deteksi loop rekursif untuk mengidentifikasi pola-pola ini.



Anda dapat menghindari loop ini dengan memastikan bahwa fungsi Lambda menulis ke sumber daya yang tidak sama dengan sumber daya yang dikonsumsi. Jika Anda harus mempublikasikan data kembali ke sumber daya konsumsi, pastikan bahwa data baru tidak memicu peristiwa yang sama. Atau, gunakan pemfilteran acara. Misalnya, berikut adalah dua solusi yang diusulkan untuk loop tak terbatas dengan sumber daya S3 dan DynamoDB:

- Jika Anda menulis kembali ke bucket S3 yang sama, gunakan awalan atau akhiran yang berbeda dari pemicu peristiwa.
- Jika Anda menulis item ke tabel DynamoDB yang sama, sertakan atribut yang dapat difilter oleh fungsi Lambda yang memakan. Jika Lambda menemukan atribut, itu tidak akan menghasilkan pemanggilan lain.

Umum: Ketidaktersediaan layanan hilir

Masalah: Layanan hilir yang diandalkan fungsi Lambda Anda tidak tersedia

Untuk fungsi Lambda yang memanggil titik akhir pihak ketiga atau sumber daya hilir lainnya, pastikan bahwa mereka dapat menangani kesalahan layanan dan batas waktu. Sumber daya hilir ini dapat memiliki waktu respons yang bervariasi, atau menjadi tidak tersedia karena gangguan layanan.

Bergantung pada implementasinya, kesalahan hilir ini mungkin muncul sebagai batas waktu atau pengecualian Lambda jika respons kesalahan layanan tidak ditangani dalam kode fungsi.

Kapan saja suatu fungsi bergantung pada layanan hilir, seperti panggilan API, menerapkan penanganan kesalahan yang sesuai, dan coba lagi logika. Untuk layanan penting, fungsi Lambda harus mempublikasikan metrik atau log ke. CloudWatch Misalnya, jika API pembayaran pihak ketiga tidak tersedia, fungsi Lambda Anda dapat mencatat informasi ini. Anda kemudian dapat mengatur CloudWatch alarm untuk mengirim pemberitahuan yang terkait dengan kesalahan ini.

Karena Lambda dapat menskalakan dengan cepat, layanan hilir non-server mungkin kesulitan menangani lonjakan lalu lintas. Ada tiga pendekatan umum untuk menangani ini:

- Caching — Pertimbangkan untuk menyimpan hasil dari nilai yang dikembalikan oleh layanan pihak ketiga jika tidak sering berubah. Anda dapat menyimpan nilai-nilai ini dalam variabel global dalam fungsi Anda, atau layanan lain. Misalnya, hasil untuk kueri daftar produk dari instans Amazon RDS dapat disimpan untuk jangka waktu tertentu dalam fungsi untuk mencegah kueri yang berlebihan.
- Antrian — Saat menyimpan atau memperbarui data, tambahkan antrean Amazon SQS antara fungsi Lambda dan sumber daya. Antrian tahan lama mempertahankan data sementara layanan hilir memproses pesan.
- Proxy — Di mana koneksi berumur panjang biasanya digunakan, seperti untuk instans Amazon RDS, gunakan lapisan proxy untuk mengumpulkan dan menggunakan kembali koneksi tersebut. Untuk database relasional, Amazon RDS Proxy adalah layanan yang dirancang untuk membantu meningkatkan skalabilitas dan ketahanan dalam aplikasi. Lambda-based

AWS SDK: Versi dan pembaruan

Masalah: AWS SDK yang disertakan pada runtime bukanlah versi terbaru

Masalah: AWS SDK yang disertakan pada runtime diperbarui secara otomatis

Runtime untuk bahasa yang ditafsirkan menyertakan versi SDK AWS . Lambda memperbarui runtime ini secara berkala untuk menggunakan versi SDK terbaru. Untuk menemukan versi SDK yang disertakan dalam runtime Anda, lihat bagian berikut:

- Runtime termasuk versi SDK () Node.js
- Runtime termasuk versi SDK (Python)
- Runtime termasuk versi SDK (Ruby)

Untuk menggunakan versi AWS SDK yang lebih baru, atau untuk mengunci fungsi Anda ke versi tertentu, Anda dapat menggabungkan pustaka dengan kode fungsi Anda, atau membuat lapisan Lambda. Untuk perincian tentang pembuatan paket deployment dengan dependensi, lihat topik berikut:

Node.js

- Menyebarkan Fungsi Acara Node.js Lambda dengan arsip file.zip

Python

- Bekerja dengan arsip file.zip untuk Fungsi Acara Python Lambda

Ruby

- Menyebarkan Fungsi Acara Ruby Lambda dengan arsip file.zip

Java

- Menyebarkan Fungsi Acara Lambda Java dengan arsip file.zip atau JAR

Go

- Terapkan Fungsi Acara Go Lambda dengan arsip file.zip

C#

- Bangun dan terapkan Fungsi Acara C# Lambda dengan arsip file.zip

PowerShell

- Menyebarkan Fungsi Acara PowerShell Lambda dengan arsip file.zip

Python: Pustaka memuat dengan tidak benar

Masalah: (Python) Beberapa pustaka tidak memuat dengan benar dari paket deployment

Pustaka dengan modul ekstensi yang tertulis dalam C atau C++ harus disusun dalam lingkungan dengan arsitektur prosesor yang sama dengan Lambda (Amazon Linux). Untuk informasi selengkapnya, lihat Bekerja dengan arsip file.zip untuk Fungsi Acara Python Lambda.

Java: Fungsi Anda membutuhkan waktu lebih lama untuk memproses peristiwa setelah memperbarui ke Java 17 dari Java 11

Masalah: (Java) Fungsi Anda membutuhkan waktu lebih lama untuk memproses peristiwa setelah memperbarui ke Java 17 dari Java 11

Tune compiler Anda menggunakan `JAVA_TOOL_OPTIONS` parameter. Runtime Lambda untuk Java 17 dan versi Java yang lebih baru mengubah opsi kompiler default. Perubahan ini meningkatkan waktu mulai dingin untuk fungsi berumur pendek, tetapi perilaku sebelumnya lebih cocok untuk fungsi komputasi yang intensif dan berjalan lebih lama. Setel `JAVA_TOOL_OPTIONS -XX:-TieredCompilation` untuk kembali ke perilaku Java 11. Untuk informasi tentang parameter `JAVA_TOOL_OPTIONS`, lihat the section called “Memahami variabel lingkungan JAVA_TOOL_OPTIONS”.

Kafka: Penanganan kesalahan dan coba lagi masalah konfigurasi

Masalah: Pemetaan sumber acara Kafka gagal mengonfigurasi pengaturan coba lagi atau tujuan yang gagal

Konfigurasi coba ulang Kafka dan tujuan saat gagal hanya tersedia untuk pemetaan sumber peristiwa dengan mode yang disediakan diaktifkan. Pastikan Anda telah mengonfigurasi `MinimumPollers` `ProvisionedPollerConfig` sebelum mencoba mengatur konfigurasi coba lagi.

Kesalahan konfigurasi umum:

- Percobaan ulang tak terbatas dengan batch bisect — Anda tidak dapat mengaktifkan `BisectBatchOnFunctionError` kapan `MaximumRetryAttempts` disetel ke -1 (tak terbatas). Tetapkan batas coba lagi yang terbatas atau nonaktifkan batch bisect.
- Rekursi topik yang sama - Topik tujuan kegagalan Kafka tidak bisa sama dengan topik sumber Anda. Pilih nama topik yang berbeda untuk topik surat mati Anda.
- Format tujuan Kafka tidak valid - Gunakan **`kafka://<topic-name>`** format saat menentukan topik Kafka sebagai tujuan gagal.
- kafka: masalah WriteData izin — Pastikan peran eksekusi Anda memiliki `kafka-cluster:WriteData` izin untuk topik tujuan. Topik tidak ada pengecualian batas waktu atau menulis masalah pembatasan API mungkin memerlukan peningkatan batas akun.

Memecahkan masalah pemetaan sumber acara di Lambda

Masalah di Lambda yang berhubungan dengan pemetaan sumber peristiwa bisa lebih kompleks karena melibatkan debugging di beberapa layanan. Selain itu, perilaku sumber peristiwa dapat berbeda berdasarkan sumber peristiwa yang tepat yang digunakan. Bagian ini mencantumkan masalah umum yang melibatkan pemetaan sumber peristiwa, dan memberikan panduan tentang cara mengidentifikasi dan memecahkan masalah mereka.

> **Note**
>
> Bagian ini menggunakan sumber peristiwa Amazon SQS untuk ilustrasi, tetapi prinsipnya berlaku untuk pemetaan sumber peristiwa lain yang mengantrekan pesan untuk Fungsi Peristiwa Lambda.

Mengidentifikasi dan mengelola throttling

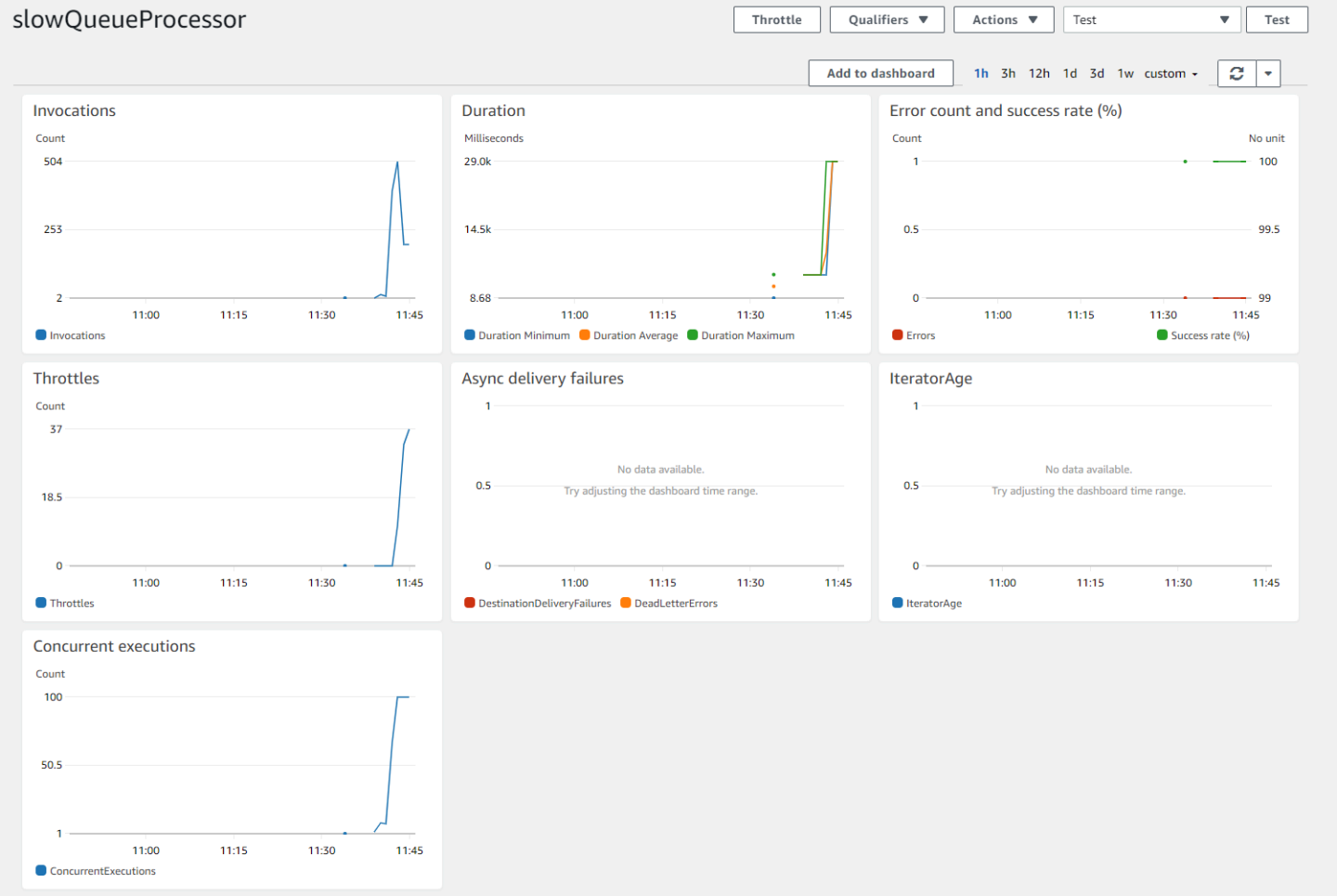
Di Lambda, pelambatan terjadi ketika Anda mencapai batas konkurensi fungsi atau akun Anda. Perhatikan contoh berikut, di mana ada fungsi Lambda yang membaca pesan dari antrian Amazon SQS. Fungsi Lambda ini mensimulasikan pemanggilan 30 detik, dan memiliki ukuran batch 1. Ini berarti bahwa fungsi hanya memproses 1 pesan setiap 30 detik:

```
const doWork = (ms) => new Promise(resolve => setTimeout(resolve, ms))

exports.handler = async (event) => {
    await doWork(30000)

}
```

Dengan waktu doa yang begitu lama, pesan mulai tiba dalam antrian lebih cepat daripada yang diproses. Jika konkurensi tanpa syarat akun Anda adalah 100, Lambda menskalakan hingga 100 eksekusi bersamaan, dan kemudian terjadi pembatasan. Anda dapat melihat pola ini di CloudWatch metrik untuk fungsi:



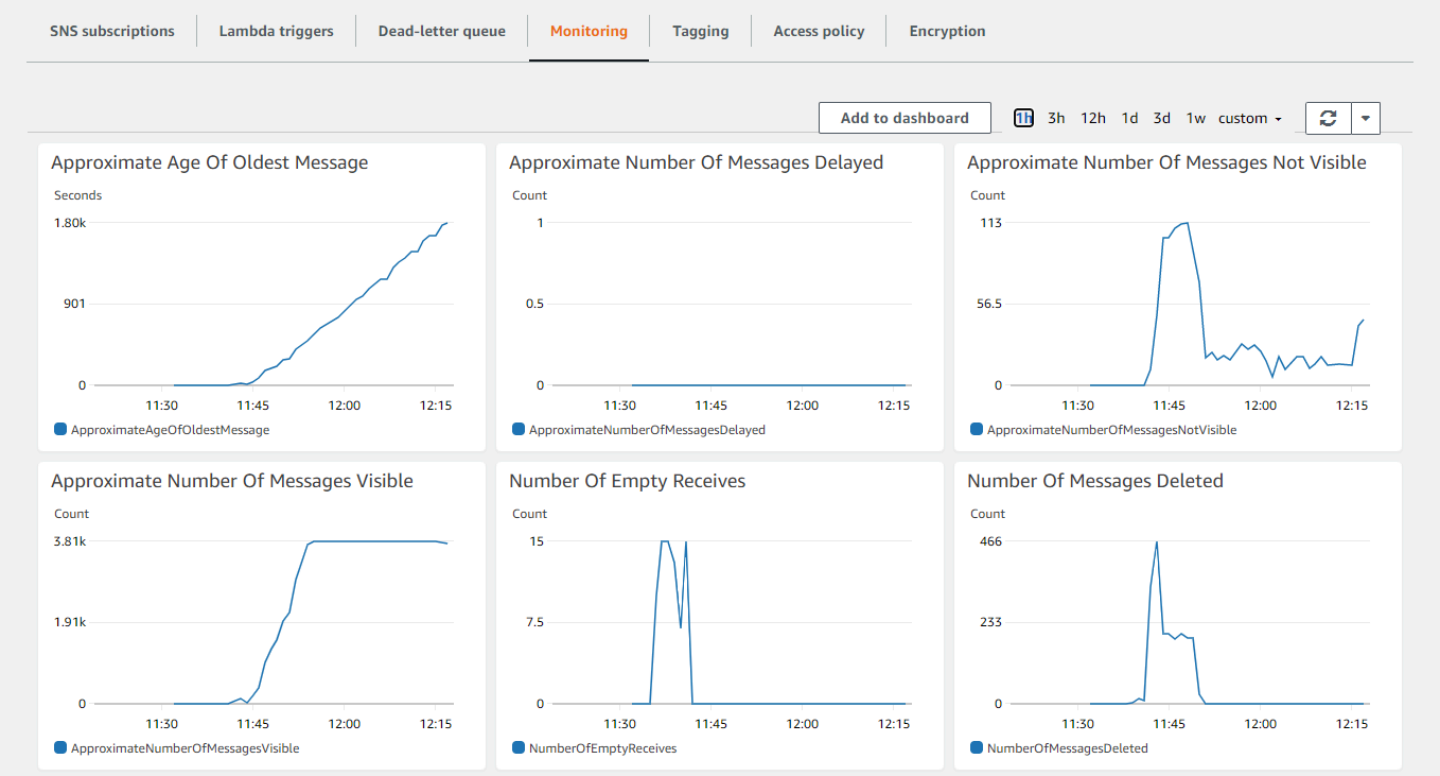
CloudWatch metrik untuk fungsi tidak menunjukkan kesalahan, tetapi bagan eksekusi bersamaan menunjukkan bahwa konkurensi maksimum 100 tercapai. Akibatnya, bagan Throttles menunjukkan throttling di tempatnya.

Anda dapat mendeteksi pelambatan dengan CloudWatch alarm, dan mengatur alarm kapan saja metrik pelambatan untuk suatu fungsi lebih besar dari 0. Setelah mengidentifikasi masalah pelambatan, Anda memiliki beberapa opsi untuk resolusi:

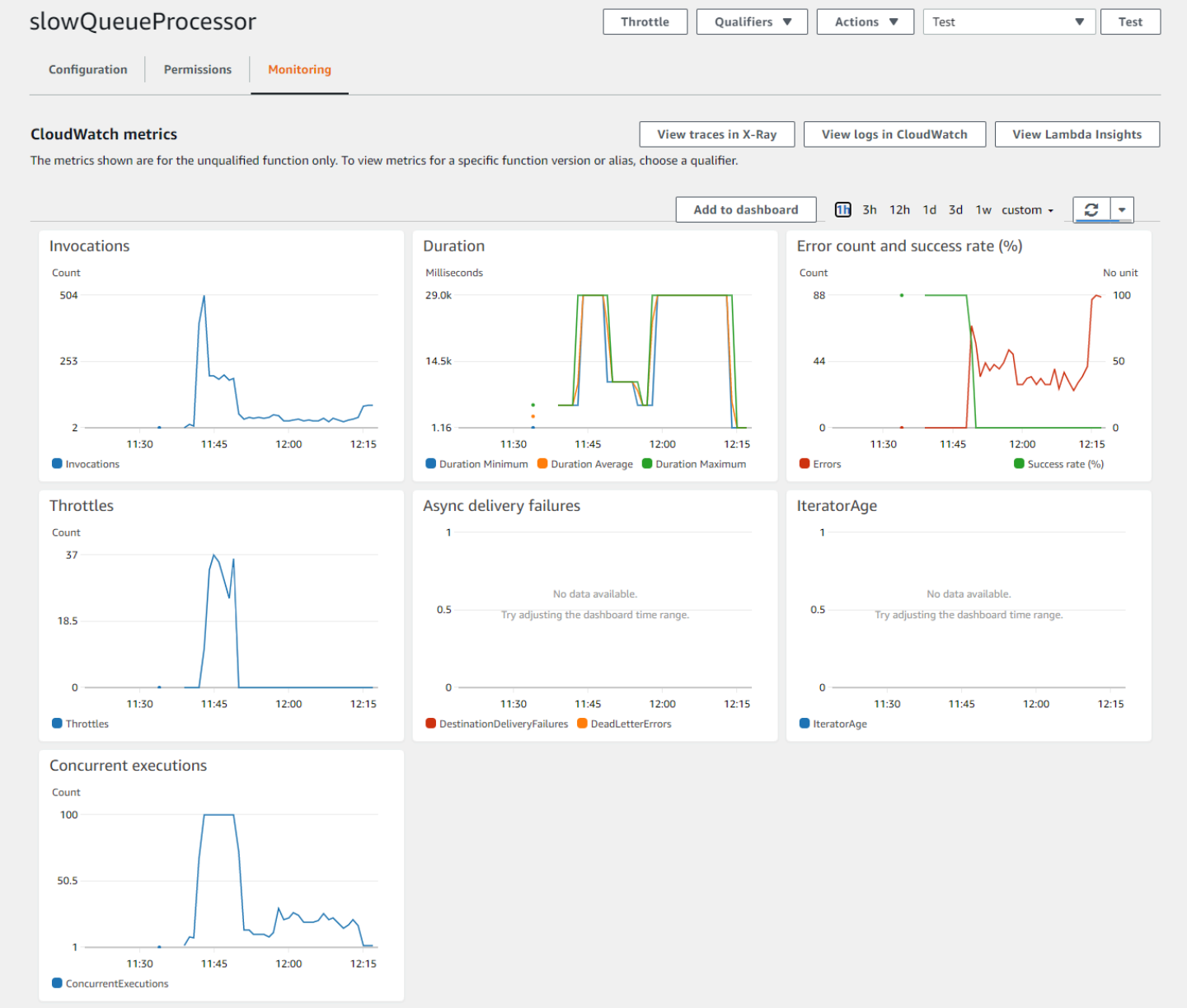
- Minta peningkatan konkurensi dari AWS Support di Wilayah ini.
- Identifikasi masalah kinerja dalam fungsi untuk meningkatkan kecepatan pemrosesan dan karenanya meningkatkan throughput.
- Tingkatkan ukuran batch fungsi, sehingga lebih banyak pesan diproses oleh setiap pemanggilan.

Kesalahan dalam fungsi pemrosesan

Jika fungsi pemrosesan melempar kesalahan, Lambda mengembalikan pesan ke antrian SQS. Lambda mencegah fungsi Anda dari penskalaan untuk mencegah kesalahan pada skala. Metrik SQS berikut CloudWatch menunjukkan masalah dengan pemrosesan antrian:

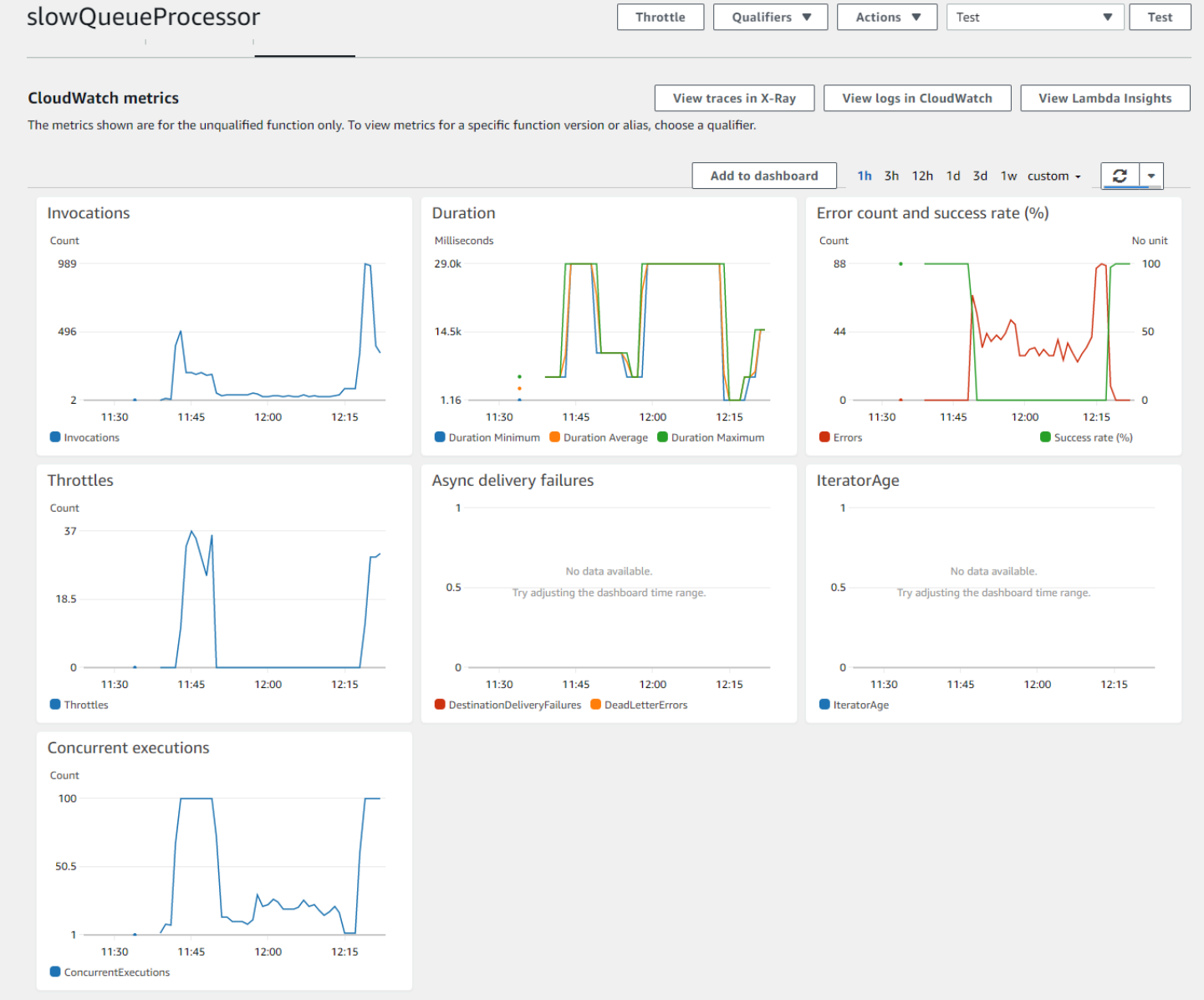


Secara khusus, usia pesan tertua dan jumlah pesan yang terlihat meningkat, sementara tidak ada pesan yang dihapus. Antrian terus bertambah tetapi pesan tidak diproses. CloudWatch Metrik untuk pemrosesan fungsi Lambda juga menunjukkan bahwa ada masalah:



Metrik jumlah kesalahan tidak nol dan berkembang, sementara eksekusi bersamaan telah berkurang dan pelambatan telah berhenti. Ini menunjukkan bahwa Lambda telah berhenti meningkatkan fungsi Anda karena kesalahan. CloudWatch Log untuk fungsi memberikan rincian jenis kesalahan.

Anda dapat mengatasi masalah ini dengan mengidentifikasi fungsi yang menyebabkan kesalahan, lalu menemukan dan menyelesaikan kesalahan. Setelah Anda memperbaiki kesalahan dan menerapkan kode fungsi baru, CloudWatch metrik akan menunjukkan pemulihan pemrosesan:



Di sini, metrik jumlah kesalahan turun menjadi nol dan metrik tingkat Sukses kembali ke 100%. Lambda mulai meningkatkan fungsi lagi, seperti yang ditunjukkan dalam grafik eksekusi Bersamaan.

Mengidentifikasi dan menangani tekanan balik

Jika produser acara secara konsisten menghasilkan pesan untuk antrian SQS lebih cepat daripada fungsi Lambda yang dapat memprosesnya, tekanan balik terjadi. Dalam hal ini, pemantauan SQS harus menunjukkan usia pesan tertua yang tumbuh secara linier, bersama dengan perkiraan jumlah pesan yang terlihat. Anda dapat mendeteksi tekanan balik dalam antrian menggunakan alarm. CloudWatch

Langkah-langkah untuk mengatasi tekanan balik tergantung pada beban kerja Anda. Jika tujuan utamanya adalah untuk meningkatkan kemampuan pemrosesan dan throughput oleh fungsi Lambda, Anda memiliki beberapa opsi:

- Minta peningkatan melalui konsol Service Quotas jika fungsi Anda mencapai kuota konkurensi akun Anda.
- Aktifkan mode yang disediakan untuk menskalakan pemetaan sumber peristiwa ke konkurensi yang lebih tinggi. Untuk informasi selengkapnya, lihat Menggunakan mode yang disediakan dengan pemetaan sumber peristiwa Amazon SQS.
- Tingkatkan ukuran batch fungsi, sehingga lebih banyak pesan diproses oleh setiap pemanggilan.

Menyelesaikan masalah jaringan di Lambda

Secara default, Lambda menjalankan fungsi Anda di virtual private cloud (VPC) dengan konektivitas ke layanan AWS dan internet. Untuk mengakses sumber daya jaringan lokal, Anda dapat mengonfigurasi fungsi Anda untuk terhubung ke VPC di akun Anda. Saat menggunakan fitur ini, Anda mengelola akses internet dan konektivitas jaringan Lambda Event Function dengan sumber daya Amazon Virtual Private Cloud (Amazon VPC).

Kesalahan konektivitas jaringan dapat diakibatkan oleh masalah dengan konfigurasi perutean VPC Anda, aturan grup keamanan, izin peran AWS Identity and Access Management (IAM), atau terjemahan alamat jaringan (NAT), atau dari ketersediaan sumber daya seperti alamat IP atau antarmuka jaringan. Bergantung pada masalahnya, Anda mungkin melihat kesalahan atau batas waktu tertentu jika permintaan tidak dapat mencapai tujuannya.

Topik

- VPC: Fungsi kehilangan akses internet atau waktu habis
- VPC: Koneksi TCP atau UDP sebentar-sebentar gagal
- VPC: Fungsi membutuhkan akses ke Layanan AWS tanpa menggunakan internet
- VPC: Batas antarmuka jaringan elastis tercapai
- EC2: Antarmuka jaringan elastis dengan jenis “lambda”
- DNS: Gagal terhubung ke host dengan UNKNOWNHOSTEXCEPTION

VPC: Fungsi kehilangan akses internet atau waktu habis

Masalah: Fungsi Lambda Anda kehilangan akses internet setelah tersambung ke VPC.

Kesalahan: Kesalahan: connect ETIMEDOUT 176.32.98.189:443

Kesalahan: Kesalahan: Batas waktu tugas berakhir setelah 10.00 detik

Kesalahan: ReadTimeoutError: Baca waktunya habis. (baca batas waktu = 15)

Saat Anda menghubungkan fungsi ke VPC, semua permintaan keluar melalui VPC. Untuk terhubung ke internet, konfigurasikan VPC Anda agar mengirim lalu lintas keluar dari subnet fungsi ke gateway NAT di subnet publik. Untuk informasi lebih lanjut dan contoh konfigurasi VPC, lihat the section called "Akses internet untuk fungsi VPC".

Jika beberapa koneksi TCP Anda habis waktu, lihat the section called "VPC: Koneksi TCP atau UDP sebentar-sebentar gagal" apakah subnet Anda menggunakan daftar kontrol akses jaringan (NACL). Jika tidak, ini kemungkinan karena fragmentasi paket. Fungsi Lambda tidak dapat menangani permintaan TCP terfragmentasi yang masuk, karena Lambda tidak mendukung fragmentasi IP untuk TCP atau ICMP.

VPC: Koneksi TCP atau UDP sebentar-sebentar gagal

> **Note**
>
> Masalah ini hanya berlaku jika subnet Anda menggunakan daftar kontrol akses jaringan (ACL). ACL jaringan tidak diperlukan untuk Lambda untuk terhubung ke subnet Anda.

Masalah: Lambda sebentar-sebentar kehilangan koneksi ke subnet VPC Anda, yang telah Anda konfigurasi untuk daftar kontrol akses jaringan (ACL).

Untuk fungsi VPC-enabled Lambda, AWS buat ENI hyperplane di akun pelanggan, dan gunakan port singkat untuk `65535` menghubungkan Lambda `1024` ke VPC pelanggan. Jika Anda menggunakan ACL jaringan di subnet target, Anda harus mengizinkan rentang port `1024` `65535` untuk TCP dan UDP. Tidak mengizinkan rentang port penuh ini dapat menyebabkan kegagalan koneksi intermiten.

VPC: Fungsi membutuhkan akses ke Layanan AWS tanpa menggunakan internet

Masalah: Fungsi Lambda Anda membutuhkan akses Layanan AWS tanpa menggunakan internet.

Untuk menghubungkan fungsi Layanan AWS dari subnet pribadi tanpa akses internet, gunakan titik akhir VPC.

VPC: Batas antarmuka jaringan elastis tercapai

Kesalahan: ENILimitReachedException: Batas antarmuka elastic network tercapai untuk VPC fungsi.

Saat Anda menghubungkan fungsi Lambda ke VPC, Lambda membuat elastic network interface untuk setiap kombinasi subnet dan grup keamanan yang terpasang pada fungsi tersebut. Kuota layanan default adalah 3.000 antarmuka jaringan per VPC. Untuk meminta peningkatan kuota, gunakan konsol Service Quotas.

EC2: Antarmuka jaringan elastis dengan jenis “lambda”

Kode Kesalahan: Client.OperationNotPermitted

Pesan galat: Grup keamanan tidak dapat dimodifikasi untuk jenis antarmuka ini

Anda menerima kesalahan ini jika Anda mencoba memodifikasi elastic network interface (ENI) yang dikelola oleh Lambda. `ModifyNetworkInterfaceAttribute`Ini tidak termasuk dalam API Lambda untuk operasi pembaruan pada antarmuka jaringan elastis yang dibuat oleh Lambda.

DNS: Gagal terhubung ke host dengan UNKNOWNHOSTEXCEPTION

Pesan Kesalahan: UNKNOWNHOSTEXCEPTION

Fungsi Lambda mendukung maksimum 20 koneksi TCP bersamaan untuk resolusi DNS. Fungsi Anda mungkin melelahkan batas itu. Permintaan DNS yang paling umum dilakukan melalui UDP. Jika fungsi Anda hanya membuat koneksi DNS UDP, ini tidak mungkin menjadi masalah Anda. Kesalahan ini biasanya terjadi karena kesalahan konfigurasi atau infrastruktur yang terdegradasi, jadi sebelum memeriksa lalu lintas DNS Anda secara mendalam, konfirmasikan bahwa infrastruktur DNS Anda dikonfigurasi dan sehat dengan benar dan bahwa fungsi Lambda Anda mengacu pada host yang ditentukan dalam DNS.

Jika Anda mendiagnosis masalah terkait dengan maksimum koneksi TCP, perhatikan bahwa Anda tidak dapat meminta peningkatan batas ini. Jika fungsi Lambda Anda kembali ke DNS TCP karena muatan DNS yang besar, konfirmasikan bahwa solusi Anda menggunakan pustaka yang mendukung EDNS. Untuk informasi lebih lanjut tentang EDNS, lihat standar RFC 6891. Jika muatan DNS Anda secara konsisten melebihi ukuran maksimal EDNS, solusi Anda mungkin masih menghabiskan batas DNS TCP.

Praktik terbaik untuk bekerja dengan AWS Lambda Fungsi Acara

Berikut adalah praktik terbaik yang direkomendasikan untuk menggunakan AWS Lambda:

Topik

- Kode Fungsi Acara
- Konfigurasi Fungsi Acara
- Skalabilitas Fungsi Acara
- Metrik dan alarm
- Bekerja dengan stream
- Praktik terbaik keamanan

Kode Fungsi Acara

Manfaatkan penggunaan kembali lingkungan eksekusi untuk meningkatkan kinerja fungsi Anda. Inisialisasi klien SDK dan koneksi basis data di luar fungsi handler, dan lakukan caching aset statis secara lokal di direktori `/tmp`. Invokasi selanjutnya yang diproses oleh instans yang sama dari fungsi Anda dapat menggunakan kembali sumber daya ini. Ini menghemat biaya dengan mengurangi waktu pengoperasian fungsi.

Untuk menghindari potensi kebocoran data di seluruh invokasi, jangan menggunakan lingkungan eksekusi untuk menyimpan data pengguna, peristiwa, atau informasi lainnya implikasi keamanan. Jika fungsi Anda bergantung pada status yang dapat disenyapkan yang tidak dapat disimpan dalam memori di dalam handler, pertimbangkan untuk membuat fungsi terpisah atau versi terpisah dari fungsi untuk setiap pengguna.

Gunakan arahan keep-alive untuk mempertahankan koneksi yang persisten. Lambda membersihkan koneksi idle dari waktu ke waktu. Mencoba menggunakan ulang koneksi idle saat mengidentifikasi suatu fungsi akan menyebabkan kesalahan koneksi. Untuk mempertahankan koneksi yang persisten, gunakan arahan tetap aktif yang berkaitan dengan runtime Anda. Sebagai contoh, lihat Menggunakan Kembali Koneksi dengan Keep-Alive in Node.js.

Gunakan variabel lingkungan untuk meneruskan parameter operasional ke fungsi Anda. Misalnya, jika Anda ingin menulis ke bucket Amazon S3 alih-alih melakukan hard-coding nama bucket yang Anda tulis, konfigurasikan nama bucket sebagai variabel lingkungan.

Hindari menggunakan pemanggilan rekursif dalam fungsi Lambda Anda, di mana fungsi memanggil dirinya sendiri atau memulai proses yang dapat memanggil fungsi lagi. Hal ini dapat menyebabkan volume invokasi fungsi yang tidak diinginkan dan peningkatan biaya. Jika Anda melihat volume pemanggilan yang tidak diinginkan, setel konkurensi fungsi cadangan untuk `0` segera membatasi semua pemanggilan ke fungsi, saat Anda memperbarui kode.

Jangan gunakan API non-publik yang tidak terdokumentasi dalam kode fungsi Lambda Anda. Untuk runtime AWS Lambda terkelola, Lambda secara berkala menerapkan pembaruan keamanan dan fungsional ke API internal Lambda. Pembaruan API internal ini mungkin tidak kompatibel ke belakang, yang menyebabkan konsekuensi yang tidak diinginkan seperti kegagalan pemanggilan jika fungsi Anda memiliki ketergantungan pada API non-publik ini. Lihat referensi API untuk daftar API yang tersedia untuk umum.

Tulis kode idempoten. Menulis kode idempoten untuk fungsi Anda memastikan bahwa peristiwa duplikat ditangani dengan cara yang sama. Kode Anda harus memvalidasi peristiwa dengan benar dan menangani peristiwa duplikat dengan anggun. Untuk informasi selengkapnya, lihat Bagaimana cara membuat fungsi Lambda saya idempoten? .

> **Note**
>
> Anda dapat menggunakan Powertools AWS Lambda untuk membuat fungsi idempoten. Untuk informasi lebih lanjut, lihat:
>
> - Python - utilitas Idempotency
> - TypeScript - Utilitas idempotensi
> - Java - utilitas Idempotency
> - .NET - Idempotency utilitas

Untuk praktik terbaik kode khusus bahasa, lihat bagian berikut:

- Praktik terbaik kode untuk Fungsi Node.js Acara Lambda
- Praktik terbaik kode untuk Fungsi TypeScript Acara Lambda
- Praktik terbaik kode untuk Fungsi Acara Lambda Python
- Praktik terbaik kode untuk Fungsi Acara Ruby Lambda
- Praktik terbaik kode untuk Fungsi Acara Java Lambda
- Praktik terbaik kode untuk Fungsi Acara Go Lambda
- Praktik terbaik kode untuk Fungsi Acara C# Lambda
- Praktik terbaik kode untuk Fungsi Acara Lambda Rust

Konfigurasi Fungsi Acara

Pengujian kinerja Fungsi Peristiwa Lambda Anda adalah bagian penting dalam memastikan Anda memilih konfigurasi ukuran memori yang optimal. Peningkatan ukuran memori apa pun memicu peningkatan CPU yang setara yang tersedia untuk fungsi Anda. Penggunaan memori untuk fungsi Anda ditentukan per pemanggilan dan dapat dilihat di Amazon. CloudWatch Pada setiap pemanggilan, `REPORT:` entri dibuat, seperti yang ditunjukkan di bawah ini:

```
REPORT RequestId: 3604209a-e9a3-11e6-939a-754dd98c7be3 Duration: 12.34 ms Billed
 Duration: 100 ms Memory Size: 128 MB Max Memory Used: 18 MB
```

Dengan menganalisis bidang `Max Memory Used:`, Anda dapat menentukan apakah fungsi Anda perlu memori lebih besar atau jika ukuran memori fungsi Anda disediakan secara berlebihan.

Untuk menemukan konfigurasi memori yang tepat untuk fungsi Anda, sebaiknya gunakan proyek AWS Lambda Power Tuning open source. Untuk informasi selengkapnya, lihat AWS Lambda Power Tuning aktif. GitHub

Untuk mengoptimalkan kinerja fungsi, kami juga merekomendasikan penerapan pustaka yang dapat menggunakan Advanced Vector Extensions 2 (AVX2). Dengan ini, Anda dapat memproses beban kerja yang menuntut, termasuk inferensi pembelajaran mesin, pemrosesan media, komputasi kinerja tinggi (HPC), simulasi ilmiah, dan pemodelan keuangan. Untuk informasi selengkapnya, lihat Membuat AWS Lambda fungsi yang lebih cepat dengan AVX2.

Uji beban Fungsi Peristiwa Lambda Anda untuk menentukan nilai batas waktu optimal. Penting untuk menganalisis berapa lama fungsi Anda berjalan sehingga Anda dapat menentukan masalah dengan layanan ketergantungan dengan lebih baik yang mungkin meningkatkan konkurensi fungsi di luar apa yang Anda harapkan. Ini sangat penting ketika Fungsi Peristiwa Lambda Anda membuat panggilan jaringan ke sumber daya yang mungkin tidak menangani penskalaan Lambda. Untuk informasi selengkapnya tentang pengujian beban aplikasi Anda, lihat Pengujian Beban Terdistribusi pada AWS.

Gunakan izin yang paling membatasi saat menyetel kebijakan IAM. Pahami sumber daya dan operasi yang dibutuhkan Fungsi Peristiwa Lambda Anda, dan batasi peran eksekusi untuk izin ini. Untuk informasi selengkapnya, lihat Mengelola izin di AWS Lambda.

Jadilah akrab denganKuota Lambda. Ukuran muatan, deskriptor file, dan ruang /tmp sering diabaikan saat menentukan batas sumber daya runtime.

Hapus Fungsi Acara Lambda yang tidak lagi Anda gunakan. Dengan melakukannya, fungsi yang tidak digunakan tidak dihitung sia-sia terhadap batas ukuran paket deployment Anda.

Jika Anda menggunakan Amazon Simple Queue Service sebagai sumber peristiwa, pastikan nilai dari waktu invokasi fungsi yang diharapkan tidak melebihi nilai Visibility Timeout di antrean. Ini berlaku baik untuk CreateFunction dan UpdateFunctionConfiguration.

- Dalam kasus CreateFunction, AWS Lambda gagal proses pembuatan fungsi.
- Dalam hal UpdateFunctionConfiguration, ini dapat menyebabkan invokasi ganda dari fungsi.

Skalabilitas Fungsi Acara

Biasakan diri dengan kendala throughput hulu dan hilir Anda. Sementara Lambda Event Functions menskalakan secara mulus dengan beban, dependensi hulu dan hilir mungkin tidak memiliki kemampuan throughput yang sama. Jika Anda perlu membatasi seberapa tinggi skala fungsi Anda, Anda dapat mengonfigurasi konkurensi cadangan pada fungsi Anda.

Bangun toleransi throttle. Jika fungsi sinkron Anda mengalami pelambatan karena lalu lintas melebihi laju penskalaan Lambda, Anda dapat menggunakan strategi berikut untuk meningkatkan toleransi throttle:

- Gunakan batas waktu, percobaan ulang, dan backoff dengan jitter. Menerapkan strategi ini memperlancar pemanggilan yang dicoba ulang, dan membantu memastikan Lambda dapat meningkatkan skala dalam hitungan detik untuk meminimalkan pelambatan pengguna akhir.
- Gunakan konkurensi yang disediakan. Konkurensi yang disediakan adalah jumlah lingkungan eksekusi pra-inisialisasi yang dialokasikan Lambda ke fungsi Anda. Lambda menangani permintaan masuk menggunakan konkurensi yang disediakan bila tersedia. Lambda juga dapat menskalakan fungsi Anda di atas dan di luar pengaturan konkurensi yang disediakan jika diperlukan. Mengkonfigurasi konkurensi yang disediakan menimbulkan biaya tambahan ke akun Anda. AWS

Metrik dan alarm

Gunakan Menggunakan CloudWatch metrik dengan Lambda dan CloudWatch Alarm alih-alih membuat atau memperbarui metrik dari dalam kode Fungsi Peristiwa Lambda Anda. Ini adalah cara yang jauh lebih efisien untuk melacak kesehatan Fungsi Acara Lambda Anda, memungkinkan Anda untuk menangkap masalah di awal proses pengembangan. Misalnya, Anda dapat mengonfigurasi alarm berdasarkan durasi yang diharapkan dari pemanggilan Fungsi Peristiwa Lambda Anda untuk mengatasi kemacetan atau latensi apa pun yang disebabkan oleh kode fungsi Anda.

Memancarkan metrik kustom secara asinkron menggunakan Embedded Metric Format (EMF). Alih-alih melakukan panggilan API sinkron CloudWatch, gunakan EMF untuk memancarkan metrik melalui log fungsi Anda. Pendekatan ini mengurangi latensi dan meningkatkan kinerja.

Utilitas Metrik di Powertools untuk AWS Lambda menangani pemformatan EMF secara otomatis. Untuk informasi selengkapnya, lihat Python, Java TypeScript, atau utilitas Metrik .NET di Powertools untuk dokumentasi. AWS Lambda Untuk informasi tentang penggunaan EMF untuk menghasilkan log format metrik, lihat Menerbitkan log dengan format metrik yang disematkan di Panduan CloudWatch Pengguna Amazon.

Gunakan logging JSON terstruktur untuk observabilitas yang lebih baik. Pencatatan terstruktur memudahkan pencarian, filter, dan analisis log fungsi Anda. Pertimbangkan untuk menggunakan utilitas Logger dari Powertools AWS Lambda untuk memformat log secara otomatis di JSON. Untuk informasi selengkapnya, lihat Python, Java TypeScript, atau utilitas .NET Logger di Powertools untuk dokumentasi. AWS Lambda

Gunakan library logging dan AWS Lambda Metrics and Dimensions untuk menangkap error aplikasi (seperti ERR, ERROR, dan WARNING).

Gunakan Deteksi Anomali AWS Biaya untuk mendeteksi aktivitas yang tidak biasa di akun Anda. Deteksi Anomali Biaya menggunakan pembelajaran mesin untuk terus memantau biaya dan penggunaan Anda sambil meminimalkan peringatan positif palsu. Deteksi Anomali Biaya menggunakan data dari AWS Cost Explorer, yang memiliki penundaan hingga 24 jam.

Akibatnya, diperlukan waktu hingga 24 jam untuk mendeteksi anomali setelah penggunaan terjadi. Untuk memulai dengan Deteksi Anomali Biaya, Anda harus terlebih dahulu mendaftar ke Cost Explorer. Kemudian, akses Deteksi Anomali Biaya.

Bekerja dengan stream

Uji dengan batch dan ukuran catatan berbeda sehingga frekuensi polling dari tiap sumber peristiwa disesuaikan dengan seberapa cepat fungsi Anda mampu menyelesaikan tugasnya. CreateEventSourceMapping BatchSize Parameter mengontrol jumlah maksimum catatan yang dapat dikirim ke fungsi Anda dengan setiap pemanggilan. Ukuran batch yang lebih besar sering kali dapat menyerap overhead secara lebih efisien di serangkaian catatan yang lebih besar, sehingga meningkatkan throughput Anda.

Secara default, Lambda memanggil fungsi Anda segera setelah catatan tersedia. Jika batch yang dibaca Lambda dari sumber peristiwa hanya memiliki satu catatan di dalamnya, Lambda hanya

mengirimkan satu catatan ke fungsi tersebut. Untuk menghindari menjalankan fungsi dengan sejumlah kecil catatan, Anda dapat memberi tahu sumber acara untuk menyangga catatan hingga 5 menit dengan mengonfigurasi jendela batching. Sebelum menjalankan fungsi, Lambda terus membaca catatan dari sumber acara hingga mengumpulkan batch penuh, jendela batching kedaluwarsa, atau batch mencapai batas muatan 6 MB. Untuk informasi selengkapnya, lihat Perilaku batching.

> **Warning**
>
> Pemetaan sumber peristiwa Lambda memproses setiap peristiwa setidaknya sekali, dan pemrosesan duplikat catatan dapat terjadi. Untuk menghindari potensi masalah yang terkait dengan duplikat peristiwa, kami sangat menyarankan agar Anda membuat kode fungsi Anda idempoten. Untuk mempelajari lebih lanjut, lihat Bagaimana cara membuat fungsi Lambda saya idempoten di Pusat Pengetahuan. AWS

Aktifkan respons batch sebagian untuk pemrosesan aliran. Saat memproses kumpulan rekaman dari aliran seperti Kinesis atau DynamoDB Streams, aktifkan respons batch paral untuk memungkinkan Lambda mencoba lagi hanya catatan yang gagal, bukan seluruh batch. Ini meningkatkan efisiensi pemrosesan dan mengurangi pemrosesan ulang yang tidak perlu. Anda dapat secara opsional menggunakan utilitas Batch dari Powertools AWS Lambda untuk menyederhanakan pola pemrosesan batch.

> **Note**
>
> Anda dapat menggunakan Powertools AWS Lambda untuk pemrosesan batch. Untuk informasi lebih lanjut, lihat:
>
> - Python - Pemrosesan Batch
> - TypeScript - Pemrosesan Batch
> - Java - Pemrosesan Batch
> - .NET - Pemrosesan Batch

Tingkatkan throughput pemrosesan aliran Kinesis dengan menambahkan pecahan. Stream Kinesis terdiri dari satu atau lebih shard. Tingkat di mana Lambda dapat membaca data dari skala Kinesis secara linier dengan jumlah pecahan.

Meningkatkan jumlah pecahan secara langsung meningkatkan jumlah pemanggilan Fungsi Peristiwa Lambda bersamaan maksimum dan dapat meningkatkan throughput pemrosesan aliran Kinesis Anda. Untuk informasi selengkapnya tentang hubungan antara pecahan dan pemanggilan fungsi, lihat. the section called “Polling dan batching stream” Jika Anda meningkatkan jumlah serpihan dalam pengaliran Kinesis, pastikan Anda sudah memilih kunci partisi yang baik (lihat Kunci Partisi) untuk data Anda, sehingga catatan terkait berakhir pada serpihan yang sama dan data Anda terdistribusi dengan baik.

Gunakan CloudWatchAmazon IteratorAge untuk menentukan apakah aliran Kinesis Anda sedang diproses. Misalnya, konfigurasikan CloudWatch alarm dengan pengaturan maksimum hingga 30000 (30 detik).

Praktik terbaik keamanan

Pantau penggunaan Anda AWS Lambda karena berkaitan dengan praktik terbaik keamanan dengan menggunakan AWS Security Hub CSPM. Security Hub CSPM menggunakan kontrol keamanan untuk mengevaluasi konfigurasi sumber daya dan standar keamanan untuk membantu Anda mematuhi berbagai kerangka kerja kepatuhan. Untuk informasi selengkapnya tentang penggunaan CSPM Security Hub untuk mengevaluasi sumber daya Lambda, lihat AWS Lambda kontrol di Panduan Pengguna. AWS Security Hub CSPM

Pantau log aktivitas jaringan Lambda menggunakan Amazon Lambda GuardDuty Protection. GuardDuty Perlindungan Lambda membantu Anda mengidentifikasi potensi ancaman keamanan saat Fungsi Peristiwa Lambda dipanggil di aplikasi Anda. Akun AWS Misalnya, jika salah satu fungsi Anda menanyakan alamat IP yang terkait dengan aktivitas terkait cryptocurrency. GuardDuty memantau log aktivitas jaringan yang dihasilkan saat Fungsi Peristiwa Lambda dipanggil. Untuk mempelajari selengkapnya, lihat Perlindungan Lambda di GuardDuty Panduan Pengguna Amazon.

Aplikasi sampel Lambda

GitHub Repositori untuk panduan ini mencakup contoh aplikasi yang menunjukkan penggunaan berbagai bahasa dan AWS layanan. Setiap aplikasi sampel menyertakan skrip untuk penyebaran dan pembersihan yang mudah serta sumber daya pendukung.

Node.js

Contoh aplikasi Lambda di Node.js

- blank-nodejs — Node.js Fungsi yang menunjukkan penggunaan logging, variabel lingkungan, AWS X-Ray tracing, lapisan, pengujian unit, dan SDK. AWS
- nodejs-apig – Fungsi dengan titik akhir API publik yang memproses peristiwa dari API Gateway dan mengembalikan respons HTTP.

Python

Aplikasi sampel Lambda di Python

- blank-python — Fungsi Python yang menunjukkan penggunaan logging, variabel lingkungan, AWS X-Ray tracing, lapisan, pengujian unit dan SDK. AWS

Ruby

Aplikasi sampel Lambda di Ruby

- blank-ruby — Fungsi Ruby yang menunjukkan penggunaan logging, variabel lingkungan, AWS X-Ray tracing, layer, pengujian unit, dan SDK. AWS
- Sampel Kode Ruby untuk AWS Lambda — Contoh kode yang ditulis dalam Ruby yang menunjukkan cara berinteraksi dengan Lambda. AWS

Java

Sampel aplikasi Lambda di Java

- example-java - Fungsi Acara Java yang menunjukkan bagaimana Anda dapat menggunakan Lambda untuk memproses pesanan. Fungsi ini menggambarkan cara mendefinisikan dan deserialisasi objek peristiwa masukan kustom, menggunakan AWS SDK, dan logging keluaran.
- java-basic - Kumpulan fungsi Java minimal dengan pengujian unit dan konfigurasi logging variabel.
- java-events - Kumpulan fungsi Java yang berisi kode kerangka untuk cara menangani peristiwa dari berbagai layanan seperti Amazon API Gateway, Amazon SQS, dan Amazon Kinesis. Fungsi-fungsi ini menggunakan versi terbaru dari pustaka aws-lambda-java-events (3.0.0 dan yang lebih baru). Contoh-contoh ini tidak memerlukan AWS SDK sebagai dependensi.

- s3-java – Fungsi Java yang memproses kejadian pemberitahuan dari Amazon S3 dan menggunakan Java Class Library (JCL) untuk membuat thumbnail dari file gambar yang diunggah.
- layer-java — Fungsi Java yang menggambarkan cara menggunakan lapisan Lambda untuk mengemas dependensi yang terpisah dari kode fungsi inti Anda.

Menjalankan kerangka kerja Java populer di Lambda

- spring-cloud-function-samples — Contoh dari Spring yang menunjukkan cara menggunakan framework Spring Cloud Function untuk membuat Lambda Event Functions. AWS
- Demo Aplikasi Boot Spring Tanpa Server - Contoh yang menunjukkan cara mengatur aplikasi Spring Boot khas dalam runtime Java yang dikelola dengan dan tanpa SnapStart, atau sebagai gambar asli GraalVM dengan runtime khusus.
- Demo Aplikasi Micronaut Tanpa Server - Contoh yang menunjukkan cara menggunakan Micronaut dalam runtime Java yang dikelola dengan dan tanpa SnapStart, atau sebagai gambar asli GraalVM dengan runtime kustom. Pelajari lebih lanjut di Micronaut/Lambda panduan.
- Demo Aplikasi Quarkus Tanpa Server - Contoh yang menunjukkan cara menggunakan Quarkus dalam runtime Java yang dikelola dengan dan tanpa SnapStart, atau sebagai gambar asli GraalVM dengan runtime kustom. Pelajari lebih lanjut di Quarkus/Lambda panduan dan Quarkus/SnapStart panduan.

Go

Lambda menyediakan aplikasi contoh berikut untuk runtime Go:

Sampel aplikasi Lambda di Go

- go-al2 - Fungsi hello world yang mengembalikan alamat IP publik. Aplikasi ini menggunakan runtime `provided.al2` khusus.
- blank-go — Fungsi Go yang menunjukkan penggunaan library Go Lambda, logging, variabel lingkungan, dan SDK. AWS Aplikasi ini menggunakan `go1.x` runtime.

C#

Aplikasi sampel Lambda dalam C#

- blank-c-sharp – Fungsi C# yang menunjukkan penggunaan pustaka NET Lambda, pencatatan, variabel lingkungan, pelacakan AWS X-Ray , uji unit, dan AWS SDK.
- blank-csharp-with-layer — Fungsi C# yang menggunakan CLI .NET untuk membuat layer yang mengemas dependensi fungsi.
- ec2-spot – Fungsi yang mengelola permintaan instans di Amazon EC2.

PowerShell

Lambda menyediakan contoh aplikasi berikut untuk: PowerShell

- blank-powershell — PowerShell Fungsi yang menunjukkan penggunaan logging, variabel lingkungan, dan SDK. AWS

Untuk men-deploy aplikasi sampel, ikuti petunjuk di file README.

Alat pengembangan untuk Lambda

Anda memiliki akses ke berbagai alat yang meningkatkan produktivitas dan kemudahan penggunaan di seluruh siklus hidup pengembangan. Bagian ini memberikan informasi tentang alat yang membantu banyak pelanggan Lambda merancang, mengembangkan, dan mengelola aplikasi mereka. Dari pengembangan lokal di IDE Anda hingga menerapkan dan mengelola aplikasi tanpa server yang kompleks, alat ini membantu Anda merampingkan alur kerja, meningkatkan kualitas kode, dan mempercepat pengembangan solusi yang kuat. Lambda-based

- Pengembangan lokal - Tulis dan uji Fungsi Acara Lambda lebih cepat di lingkungan pengembangan pilihan Anda. AWS Toolkit for VS Code memungkinkan pengembangan fungsi lokal, debugging, dan pengujian dengan kemampuan penerapan langsung ke Lambda.
- Infrastructure as Code (IAc) — Menyebarkan dan mengelola aplikasi tanpa server secara konsisten dari pengujian lokal hingga produksi. AWS SAM, AWS CDK, dan CloudFormation memungkinkan Anda menentukan dan mengelola infrastruktur tanpa server Anda melalui kode untuk penerapan yang konsisten dan dikendalikan versi.
- GitHub Tindakan - Otomatiskan penerapan Lambda langsung dari repositori kode Anda. Dengan GitHub Tindakan, Anda dapat mengatur alur kerja yang secara otomatis menerapkan

fungsi Lambda setiap kali Anda mendorong perubahan kode atau konfigurasi, sehingga menyederhanakan pipeline Anda. CI/CD

- Powertools untuk AWS Lambda - Bangun aplikasi tanpa server yang siap produksi dengan kode kustom yang lebih sedikit. Powertools untuk AWS Lambda (juga disebut sebagai Powertools AWS for) adalah toolkit pengembang open-source yang menyederhanakan penerapan praktik terbaik tanpa server seperti observabilitas, pengambilan parameter, dan idempotensi di seluruh Python, Java, dan .NET. TypeScript
- Alur kerja dan acara — Mengkoordinasikan fungsi Lambda AWS dengan layanan, API, dan sistem eksternal. Lambda menyediakan dua opsi orkestrasi: Lambda fungsi tahan lama untuk orkestrasi aplikasi-sentris menggunakan bahasa pemrograman standar dalam Lambda, dan untuk orkestrasi yang berpusat pada alur kerja dengan desain visual di berbagai layanan. AWS Step Functions Amazon EventBridge menyediakan kemampuan manajemen acara untuk arsitektur berbasis peristiwa. Untuk bantuan memilih pendekatan orkestrasi, lihat Fungsi tahan lama atau Step Functions.

Alat pengembangan lokal

Lingkungan pengembangan lokal memungkinkan Anda untuk bekerja offline dan menggunakan fitur IDE canggih sambil mengulangi dengan cepat pada fungsi Lambda Anda. Alat-alat ini membantu Anda men-debug fungsi kompleks dan berkembang di lingkungan dengan konektivitas terbatas. Mereka juga mendukung kolaborasi tim dan integrasi dengan sistem kontrol versi.

Untuk informasi selengkapnya tentang pengembangan fungsi Lambda secara lokal, lihat. Mengembangkan Fungsi Acara secara lokal Halaman ini menjelaskan cara memindahkan pengembangan fungsi Lambda dari AWS konsol ke Visual Studio Code, yang menyediakan lingkungan pengembangan yang kaya dengan fitur seperti debugging dan penyelesaian kode. Untuk melakukan transisi, Anda perlu mengatur AWS Toolkit for Visual Studio Code dan kredensialnya, setelah itu Anda dapat menggunakan fitur-fitur canggih di VS Code sambil mempertahankan kemampuan untuk menyebarkan langsung ke. AWS

Pengembangan lokal untuk Lambda menyediakan beberapa kemampuan utama:

- Gunakan integrasi Visual Studio Code dengan konsol Lambda
- Konfigurasikan lingkungan pengembangan Lambda lokal
- Fungsi debug dan uji secara lokal
- Menerapkan praktik terbaik untuk manajemen fungsi lokal

Untuk informasi selengkapnya, lihat Mengembangkan Fungsi Acara secara lokal.

Infrastruktur sebagai alat Kode (IAc)

Dengan alat Infrastructure as Code (IAc), Anda dapat menentukan dan mengelola arsitektur tanpa server menggunakan kode. Pendekatan ini membantu menjaga konsistensi di seluruh lingkungan, memberi Anda kendali atas versi infrastruktur Anda, dan mendukung DevOps praktik. IAc sangat berharga untuk mengotomatiskan penerapan, memastikan lingkungan yang konsisten, dan mengelola penyebaran multi-wilayah.

Alat dan konsep utama IAC untuk Lambda mencakup kerangka kerja untuk pembuatan template, manajemen penerapan, dan praktik terbaik untuk infrastruktur tanpa server:

- Prinsip Core IAc untuk pengembangan Lambda
- CloudFormation, AWS SAM, dan AWS CDK kemampuan
- Kriteria dan perbandingan pemilihan alat
- Praktik terbaik untuk implementasi Lambda IAc

Baik Anda bekerja secara mandiri pada proyek kecil atau sebagai bagian dari tim besar yang mengelola aplikasi tanpa server skala perusahaan, alat pengembangan dan penyebaran ini dapat membantu Anda menulis, menyebarkan, dan mengelola fungsi Lambda Anda dengan lebih efektif.

Untuk informasi selengkapnya, lihat Menggunakan Lambda dengan infrastruktur sebagai kode (IAc).

GitHub Alat tindakan

GitHub Actions menyediakan kemampuan penerapan otomatis untuk fungsi Lambda Anda langsung dari repositori kode Anda. Dengan membuat file alur kerja di repositori, Anda dapat secara otomatis menerapkan fungsi Lambda setiap kali perubahan kode atau konfigurasi didorong, merampingkan integrasi berkelanjutan dan pipeline deployment () berkelanjutan. CI/CD Tindakan fungsi Deploy Lambda menawarkan antarmuka YAMG deklaratif yang menyederhanakan proses penerapan, menangani kredensil AWS melalui OpenID Connect (OIDC), dan mendukung berbagai skenario penerapan termasuk pembaruan kode, perubahan konfigurasi, dan validasi dry run. Integrasi ini memungkinkan tim untuk mempertahankan proses penerapan yang konsisten dan otomatis saat menggunakan GitHub alur kerja yang ada.

Untuk informasi selengkapnya, lihat Menggunakan GitHub Tindakan untuk menyebarkan Fungsi Acara.

Powertools untuk AWS Lambda

Powertools for AWS adalah toolkit pengembang open-source yang membantu Anda menerapkan praktik terbaik tanpa server dengan kode kustom minimal. Tersedia untuk Python, Java, TypeScript/Node.js dan .NET, ia menyediakan fungsi utilitas, dekorator, dan middleware yang merampingkan tugas-tugas pengembangan Lambda umum. Toolkit ini mencakup fitur observabilitas bawaan seperti pencatatan terstruktur, penelusuran, dan pengumpulan metrik, seperti utilitas untuk pengambilan parameter, manajemen rahasia, dan pola idempotensi. Alat-alat ini selaras dengan praktik terbaik AWS yang dirancang dengan baik dan membantu pengembang membangun aplikasi tanpa server yang siap produksi dengan lebih efisien. Dengan mengurangi kode boilerplate dan menstandarisasi pola umum, Powertools for AWS memungkinkan tim untuk fokus pada logika bisnis sambil mempertahankan implementasi praktik terbaik tanpa server yang konsisten di seluruh aplikasi mereka.

Untuk informasi selengkapnya, lihat Powertools untuk AWS Lambda.

Alur kerja dan alat manajemen acara

Aplikasi Lambda dapat digunakan dalam orkestrasi alur kerja yang kompleks dan penanganan berbagai peristiwa. AWS menyediakan alat khusus untuk membantu Anda mengelola aspek-aspek pengembangan tanpa server ini. Pelajari tentang AWS Step Functions orkestrasi alur kerja dan EventBridge Amazon untuk manajemen acara, dan cara mengintegrasikannya dengan fungsi Lambda Anda. Alat-alat ini dapat secara signifikan meningkatkan skalabilitas dan keandalan aplikasi tanpa server Anda dengan menyediakan manajemen status yang kuat dan arsitektur berbasis peristiwa. Dengan menggunakan layanan ini, Anda dapat membangun Lambda-based solusi yang lebih canggih dan tangguh yang dapat menangani proses bisnis yang kompleks dan bereaksi terhadap berbagai peristiwa sistem dan aplikasi.

Lihat informasi yang lebih lengkap di Mengelola alur kerja dan acara Lambda.

Mengembangkan Fungsi Acara secara lokal

Anda dapat memindahkan Fungsi Acara Lambda dari konsol Lambda ke lingkungan pengembangan terintegrasi (IDE) pilihan Anda, yang menyediakan lingkungan pengembangan penuh di mana Anda dapat menggunakan opsi pengembangan lokal lainnya seperti dan. AWS SAM AWS CDK Konsol Lambda mendukung fungsi pembukaan langsung di Visual Studio Code, Kiro, atau Cursor.

Manfaat utama pembangunan lokal

Sementara konsol Lambda menyediakan cara cepat untuk mengedit dan menguji fungsi, pengembangan lokal menawarkan kemampuan yang lebih canggih:

- Fitur IDE lanjutan: Debugging, penyelesaian kode, dan alat refactoring
- Pengembangan offline: Bekerja dan uji perubahan secara lokal sebelum penerapan cloud
- Infrastruktur sebagai integrasi kode: Penggunaan tanpa batas dengan AWS SAM, AWS CDK, dan Komposer Infrastruktur
- Manajemen ketergantungan: Kontrol penuh atas dependensi fungsi

Prasyarat

Sebelum mengembangkan fungsi Lambda secara lokal menggunakan Visual Studio Code, Kiro, dan Cursor, Anda harus menginstalnya menggunakan petunjuk di bawah ini:

- Kode Visual Studio: Untuk petunjuk penginstalan, lihat Mengunduh Kode Visual Studio.
- Catatan: Untuk petunjuk pemasangan, lihat Memulai dengan Kiro.
- Kursor: Untuk petunjuk penginstalan, lihat Unduh Kursor.

Anda juga memerlukan hal berikut:

- AWS Toolkit for Visual Studio Code: Untuk petunjuk instalasi Visual Studio Code, lihat Menyiapkan AWS Toolkit for Visual Studio Code. Untuk ikhtisar, lihat AWS Toolkit for Visual Studio Code.
- AWS kredensial: Untuk informasi tentang mengonfigurasi kredensil, lihat Menyiapkan kredensil Anda. AWS
- AWS SAMCLI: Untuk petunjuk penginstalan, lihat Menginstal AWS SAMCLI.
- Docker diinstal (opsional, tetapi diperlukan untuk pengujian lokal): Untuk petunjuk instalasi, lihat Mendapatkan Docker.

> **Note**
>
> Jika Anda sudah memiliki AWS akun dan profil yang dikonfigurasi secara lokal, pastikan bahwa kebijakan AdministratorAccess terkelola ditambahkan ke AWS profil yang dikonfigurasi.

Kontrol autentikasi dan akses

Untuk mengembangkan fungsi Lambda secara lokal, Anda memerlukan AWS kredensil untuk mengakses dan mengelola AWS sumber daya dengan aman atas nama Anda, seperti yang mereka lakukan di cloud.

AWS Toolkit for Visual Studio Code mendukung metode otentikasi berikut:

- Kredensi jangka panjang pengguna IAM
- Kredensi sementara dari peran yang diasumsikan
- Federasi identitas
- AWS kredensi pengguna root akun (tidak disarankan)

Bagian ini memandu Anda untuk memperoleh dan mengonfigurasi kredensil ini menggunakan kredensil jangka panjang pengguna IAM.

Dapatkan Kredensial IAM

Jika Anda sudah memiliki pengguna IAM dengan kunci akses, siapkan ID kunci akses dan kunci akses rahasia untuk bagian berikutnya. Jika Anda tidak memiliki kunci ini, ikuti langkah-langkah berikut untuk membuatnya:

> **Note**
>
> Anda harus menggunakan ID kunci akses dan kunci akses rahasia bersama-sama untuk mengautentikasi permintaan Anda.

Untuk membuat pengguna IAM dan kunci akses:

1. Buka konsol IAM di https://console.aws.amazon.com/iam/
2. Di panel navigasi, pilih Users (Pengguna).
3. Pilih Create user (Buat pengguna).
4. Untuk Nama pengguna, masukkan nama dan pilih Berikutnya.
5. Di bawah Setel izin, pilih Lampirkan kebijakan secara langsung.
6. Pilih AdministratorAccessdan pilih Berikutnya.

7. Pilih Create user (Buat pengguna).
8. Di spanduk sukses, pilih Lihat pengguna.
9. Pilih Buat access key.
10. Untuk kasus penggunaan, pilih Kode lokal.
11. Pilih kotak centang konfirmasi dan pilih Berikutnya.
12. (Opsional) Masukkan nilai tag deskripsi.
13. Pilih Buat access key.
14. Salin kunci akses dan kunci akses rahasia Anda segera. Anda tidak akan dapat mengakses kunci akses rahasia lagi setelah Anda meninggalkan halaman ini.

> **Important**
>
> Jangan pernah membagikan kunci rahasia Anda atau berkomitmen untuk kontrol sumber. Simpan kunci ini dengan aman dan hapus saat tidak diperlukan lagi.

> **Note**
>
> Untuk informasi selengkapnya, lihat Membuat pengguna IAM di AWS akun Anda dan Mengelola kunci akses untuk pengguna IAM di Panduan Pengguna IAM.

Konfigurasi AWS kredensil menggunakan AWS Toolkit

Tabel berikut merangkum proses penyiapan kredenal yang Anda selesaikan dalam prosedur berikut.

Apa yang harus dilakukan	Kenapa?
Buka panel Masuk	Mulai otentikasi
Gunakan Command Palette, cari AWS Tambahkan Koneksi Baru	Mengakses UI masuk
Pilih IAM Credential	Gunakan kunci akses Anda untuk akses terprogram

Apa yang harus dilakukan	Kenapa?
Masukkan nama profil, kunci akses, kunci rahasia	Memberikan kredensi untuk koneksi
Lihat Pembaruan AWS Explorer	Konfirmasikan bahwa Anda terhubung

Selesaikan langkah-langkah berikut untuk mengautentikasi ke AWS akun Anda:

> **Note**
>
> Langkah-langkah berikut berlaku untuk Visual Studio Code, Kiro, dan Cursor. Karena Kiro dan Cursor dibangun di atas Visual Studio Code, aliran otentikasi sama di ketiga IDE.

1. Buka panel Masuk di IDE Anda:

   Untuk memulai proses otentikasi, pilih AWS ikon di panel navigasi kiri atau buka Palet Perintah (Cmd+Shift+P di Mac atau Ctrl+Shift+P aktif) dan cari dan pilih Tambahkan Koneksi Baru. Windows/Linux AWS
2. Pada panel masuk, pilih IAM Credentials dan pilih Continue.

   > **Note**
   >
   > Untuk melanjutkan, Anda perlu mengizinkan Ekstensi AWS IDE untuk mengakses data Anda.
3. Masukkan nama profil Anda, ID kunci akses, dan kunci akses rahasia, lalu pilih Lanjutkan.
4. Verifikasi koneksi dengan memeriksa AWS Explorer di IDE Anda untuk AWS layanan dan sumber daya Anda.

Untuk informasi tentang cara menyiapkan autentikasi dengan kredensi jangka panjang, lihat Menggunakan kredensil jangka panjang untuk mengautentikasi SDK dan alat bantu. AWS

Untuk informasi tentang mengonfigurasi autentikasi, lihat AWS kredensil IAM di Panduan Pengguna. AWS Toolkit for Visual Studio Code

Pindah dari konsol ke pengembangan lokal

> **Note**
>
> Jika Anda telah membuat perubahan di konsol, pastikan Anda tidak memiliki perubahan yang tidak diterapkan sebelum beralih ke pengembangan lokal.

Untuk memindahkan fungsi Lambda dari konsol Lambda ke IDE lokal, selesaikan langkah-langkah berikut:

1. Buka Konsol Lambda.
2. Pilih nama fungsi Anda.
3. Pilih tab Sumber kode.
4. Buka fungsi Anda di IDE pilihan Anda dengan memilih panah tarik-turun di sebelah tombol. Opsi yang didukung adalah Buka di Visual Studio Code, Buka di Kiro, dan Buka di Kursor.

   > **Note**
   >
   > Pilihan IDE Anda tetap ada. Setelah Anda memilih IDE, kunjungan berikutnya menunjukkan bahwa IDE sebagai tombol utama.

5. Saat diminta, izinkan browser Anda membuka IDE yang dipilih.

Saat Anda membuka fungsi di IDE, Lambda membuat proyek lokal dengan kode fungsi Anda di lokasi sementara yang dirancang untuk pengujian dan penerapan cepat. Ini termasuk kode fungsi, dependensi, dan struktur proyek dasar yang dapat Anda gunakan untuk pengembangan lokal.

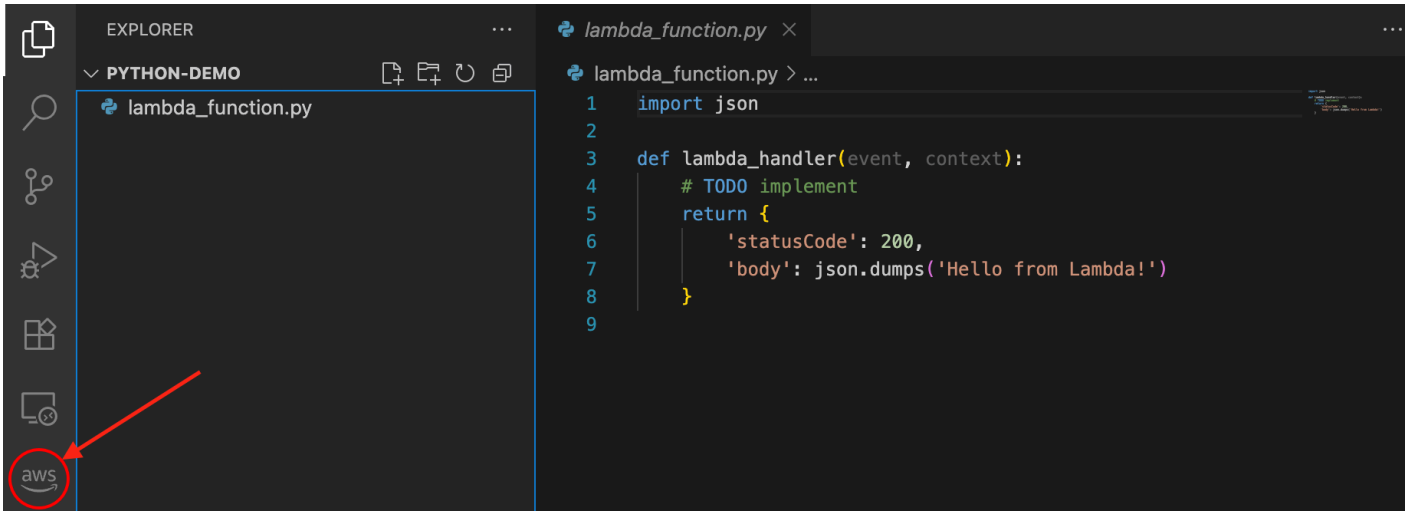
> **Note**
>
> Tombol Buka di IDE memerlukan AWS Toolkit versi 3.69.0 atau yang lebih baru. Jika Anda memiliki versi AWS Toolkit yang lebih lama diinstal, Anda mungkin melihat `Cannot open the handler` pesan. Untuk mengatasinya, perbarui AWS Toolkit Anda ke versi terbaru.

Untuk detail tentang penggunaan AWS di IDE Anda, lihat Panduan AWS Toolkit for Visual Studio Code Pengguna.

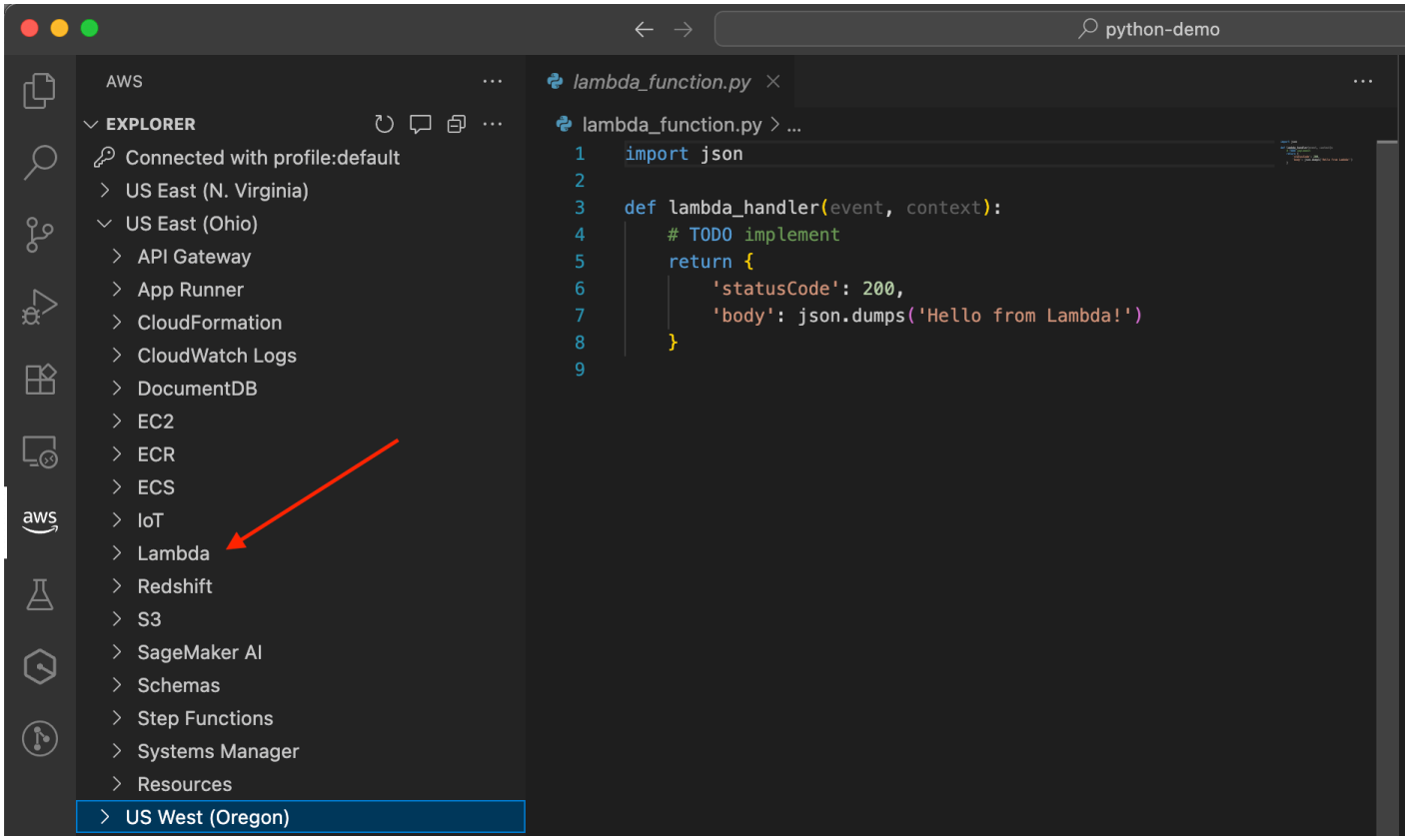
Bekerja dengan fungsi lokal

Setelah membuka fungsi Anda di IDE pilihan Anda, ikuti langkah-langkah ini untuk mengakses dan mengelola fungsi Anda. Untuk contoh di bawah ini kita telah menggunakan Visual Studio Code.

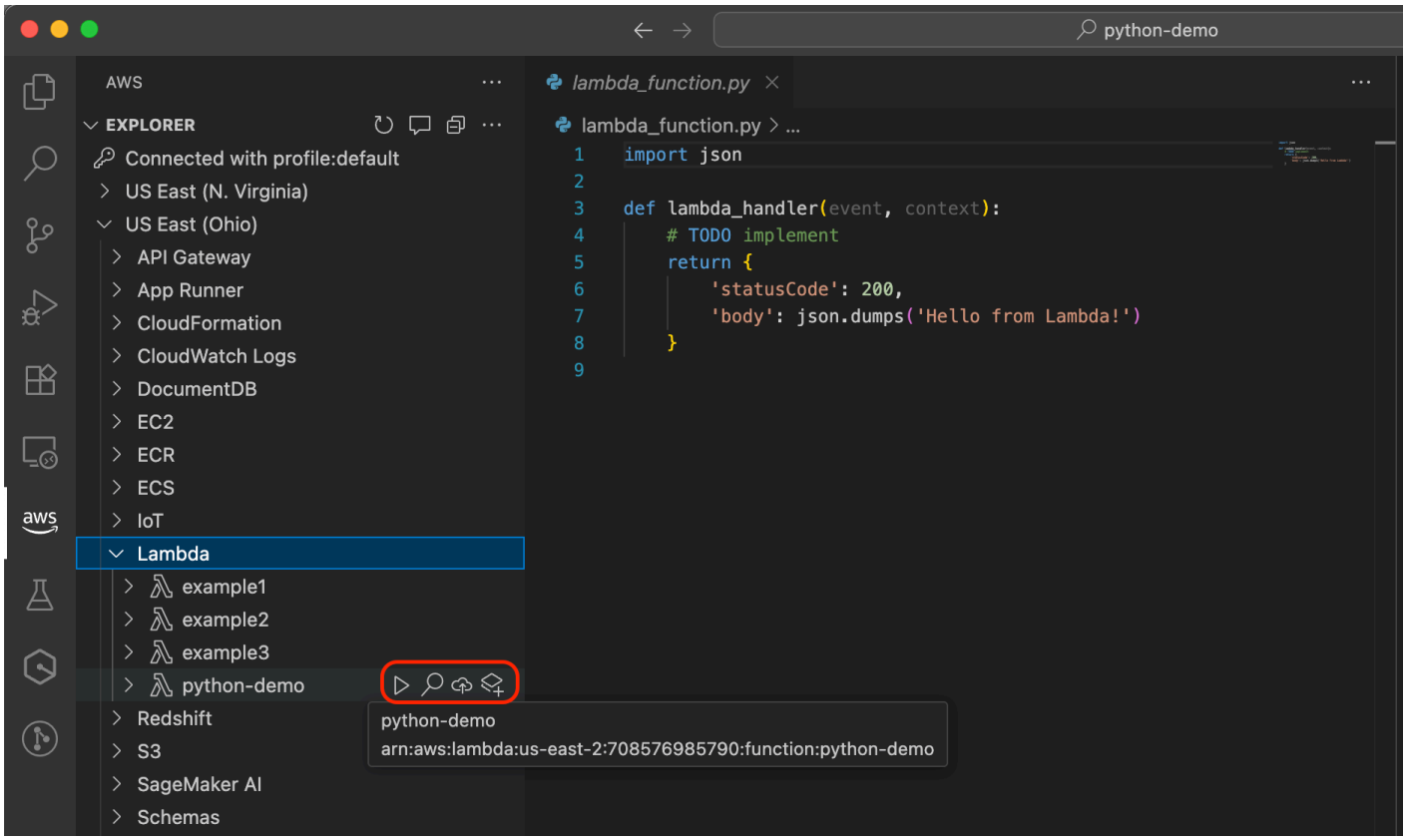
1. Pilih AWS ikon di bilah sisi untuk membuka AWS Explorer:



2. Di AWS Explorer, pilih wilayah dengan fungsi Lambda Anda:



3. Di bawah wilayah yang Anda pilih, perluas bagian Lambda untuk melihat dan mengelola fungsi Anda:



Dengan fungsi Anda dibuka di IDE Anda, Anda dapat:

- Edit kode fungsi dengan dukungan bahasa penuh dan penyelesaian kode.
- Gunakan LocalStack integrasi untuk menguji fungsi Lambda yang membuat panggilan API ke yang lain Layanan AWS selama eksekusi, seperti membaca dari tabel DynamoDB atau menulis ke bucket Amazon S3. LocalStack adalah emulator layanan cloud yang menyediakan lingkungan pengembangan lokal lengkap untuk menguji integrasi layanan. Anda juga dapat menggunakan AWS SAMCLI untuk menguji fungsi Anda dalam wadah lokal. Jika fungsi Anda membuat panggilan API ke yang lain Layanan AWS, panggilan tersebut akan mencapai AWS sumber daya nyata, bukan yang ditiru.
- Debug fungsi Anda dengan breakpoint dan inspeksi variabel. Untuk informasi selengkapnya, lihat Menjalankan dan men-debug fungsi Lambda langsung dari kode di Panduan Pengguna AWS Toolkit for Visual Studio Code .
- Terapkan kembali fungsi Anda yang diperbarui untuk AWS menggunakan ikon cloud.
- Instal dan kelola dependensi untuk fungsi Anda.

Konversikan fungsi Anda menjadi AWS SAM template dan gunakan alat IAc

Di semua IDE yang didukung (Visual Studio Code, Kiro, dan Cursor), Anda dapat mengonversi fungsi Lambda Anda ke AWS SAM template dengan memilih ikon Konversi ke AWS SAM Aplikasi di sebelah fungsi Lambda Anda. Anda diminta untuk memilih lokasi AWS SAM proyek. Setelah dipilih, fungsi Lambda Anda dikonversi ke `template.yaml` file yang disimpan di proyek baru AWS SAM Anda.

Dengan fungsi Anda dikonversi ke AWS SAM template, Anda dapat:

- Kontrol versi infrastruktur Anda
- Mengotomatiskan penerapan
- Fungsi debug jarak jauh
- Tambahkan AWS sumber daya tambahan ke aplikasi Anda
- Pertahankan lingkungan yang konsisten di seluruh siklus hidup pengembangan Anda
- Gunakan Infrastructure Composer untuk mengedit template Anda AWS SAM secara visual

Untuk informasi lebih lanjut tentang penggunaan alat IAC, lihat panduan berikut:

- Panduan AWS Serverless Application Model Pengembang
- Panduan AWS Cloud Development Kit (AWS CDK) Pengembang
- Panduan Pengembang Komposer Infrastruktur
- Panduan AWS CloudFormation Pengguna

Alat-alat ini menyediakan kemampuan tambahan untuk mendefinisikan, menguji, dan menyebarkan aplikasi tanpa server Anda.

Langkah selanjutnya

Untuk mempelajari selengkapnya tentang bekerja dengan fungsi Lambda di Visual Studio Code, lihat sumber daya berikut:

- Bekerja dengan fungsi AWS Lambda di Panduan Pengguna Kode AWS Toolkit for Visual Studio
- Bekerja dengan aplikasi tanpa server di Panduan Pengguna Kode AWS Toolkit for Visual Studio
- Infrastruktur sebagai kode dalam Panduan Pengembang Lambda

Menggunakan GitHub Tindakan untuk menyebarkan Fungsi Acara

Anda dapat menggunakan GitHub Tindakan untuk secara otomatis menerapkan Fungsi Peristiwa Lambda saat Anda mendorong perubahan kode atau konfigurasi ke repositori Anda. Tindakan Deploy Lambda Function menyediakan antarmuka YAML deklaratif dan sederhana yang menghilangkan kompleksitas langkah penerapan manual.

Contoh alur kerja

Untuk mengonfigurasi penerapan fungsi Lambda otomatis, buat file alur kerja di direktori repositori Anda: `.github/workflows/`

Example GitHub Alur kerja tindakan untuk penerapan Lambda

```
name: Deploy AWS Lambda
on:
  push:
    branches:
      - main

jobs:
  deploy:
    runs-on: ubuntu-latest
    permissions:
      id-token: write # Required for OIDC authentication
      contents: read  # Required to check out the repository
    steps:
      - uses: actions/checkout@v4

      - name: Configure AWS credentials
        uses: aws-actions/configure-aws-credentials@v4
        with:
          role-to-assume: arn:aws:iam::123456789012:role/GitHubActionRole
          aws-region: us-east-1

      - name: Deploy Lambda Function
        uses: aws-actions/aws-lambda-deploy@v1
        with:
          function-name: my-lambda-function
          code-artifacts-dir: ./dist
```

Alur kerja ini berjalan saat Anda mendorong perubahan ke `main` cabang. Ini memeriksa repositori Anda, mengonfigurasi AWS kredensil menggunakan OpenID Connect (OIDC), dan menyebarkan fungsi Anda menggunakan kode di direktori. `./dist`

Untuk contoh tambahan termasuk memperbarui konfigurasi fungsi, menerapkan melalui bucket S3, dan validasi dry run, lihat README Fungsi Lambda Deploy.

Sumber daya tambahan

- Konfigurasikan AWS Tindakan GitHub Kredensial
- Mengkonfigurasi OpenID Connect di AWS

Menggunakan Lambda dengan infrastruktur sebagai kode (IAc)

Fungsi Acara Lambda jarang berjalan secara terpisah. Sebaliknya, mereka sering membentuk bagian dari aplikasi tanpa server dengan sumber daya lain seperti database, antrian, dan penyimpanan. Dengan Infrastructure as code (IAc), Anda dapat mengotomatiskan proses penyebaran Anda untuk menyebarkan dan memperbarui seluruh aplikasi tanpa server dengan cepat dan berulang yang melibatkan banyak sumber daya terpisah. AWS Pendekatan ini mempercepat siklus pengembangan Anda, membuat manajemen konfigurasi lebih mudah, dan memastikan bahwa sumber daya Anda digunakan dengan cara yang sama setiap saat.

Alat IAc untuk Lambda

CloudFormation

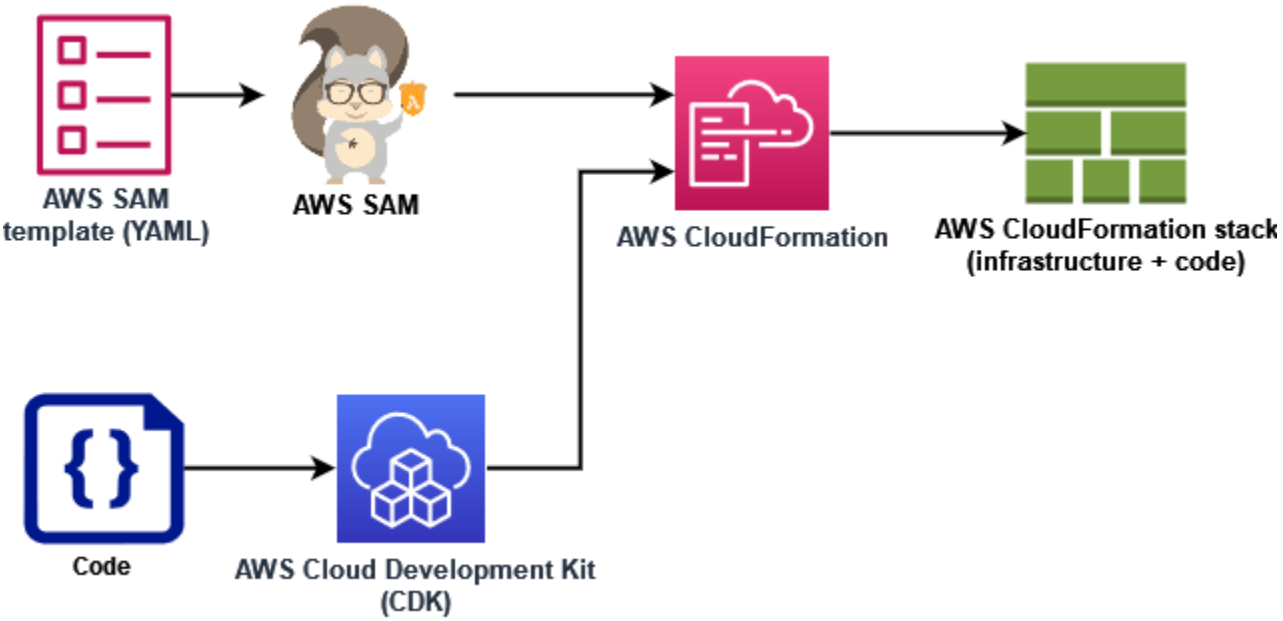
> CloudFormation adalah layanan IAc dasar dari. AWS Anda dapat menggunakan templat YAMB atau JSON untuk memodelkan dan menyediakan seluruh AWS infrastruktur Anda, termasuk fungsi Lambda. CloudFormation menangani kompleksitas membuat, memperbarui, dan menghapus sumber daya Anda AWS .

AWS Serverless Application Model (AWS SAM)

> AWS SAM adalah kerangka kerja sumber terbuka yang dibangun di atas. CloudFormation Ini menyediakan sintaks yang disederhanakan untuk mendefinisikan aplikasi tanpa server. Gunakan AWS SAM template untuk menyediakan fungsi Lambda, API, database, dan sumber peristiwa dengan cepat hanya dengan beberapa baris YAMB.

AWS Cloud Development Kit (AWS CDK)

CDK adalah pendekatan kode-pertama untuk IAc. Anda dapat menentukan Lambda-based arsitektur Anda menggunakan TypeScript,, Python JavaScript, Java, C #/.Net, atau Go. Pilih bahasa pilihan Anda dan gunakan elemen pemrograman seperti parameter, kondisional, loop, komposisi, dan pewarisan untuk menentukan hasil yang diinginkan dari infrastruktur Anda. CDK kemudian menghasilkan CloudFormation template yang mendasari untuk penerapan. Untuk contoh cara menggunakan Lambda dengan CDK, lihat. Menerapkan Fungsi Acara dengan AWS CDK



AWS juga menyediakan layanan yang dipanggil AWS Infrastructure Composer untuk mengembangkan template IAC menggunakan antarmuka grafis sederhana. Dengan Infrastructure Composer, Anda merancang arsitektur aplikasi dengan menyeret, mengelompokkan, dan menghubungkan Layanan AWS dalam kanvas visual. Infrastructure Composer kemudian membuat AWS SAM CloudFormation template atau template dari desain Anda yang dapat Anda gunakan untuk menyebarkan aplikasi Anda.

Pada the section called “Menggunakan AWS SAM dan Infrastruktur Komposer” bagian di bawah ini, Anda menggunakan Infrastructure Composer untuk mengembangkan template untuk aplikasi tanpa server berdasarkan fungsi Lambda yang ada.

Menggunakan Fungsi Acara di AWS SAM dan Komposer Infrastruktur

Dalam tutorial ini, Anda dapat mulai menggunakan IAc dengan Lambda dengan membuat template dari fungsi Lambda AWS SAM yang ada dan kemudian membangun aplikasi tanpa server di Infrastructure Composer dengan menambahkan sumber daya lainnya. AWS

Saat Anda menjalankan tutorial ini, Anda akan mempelajari beberapa konsep dasar, seperti bagaimana AWS sumber daya ditentukan AWS SAM. Anda juga akan belajar cara menggunakan Infrastructure Composer untuk membangun aplikasi tanpa server yang dapat Anda gunakan menggunakan atau. AWS SAM CloudFormation

Untuk menyelesaikan tutorial ini, Anda akan melakukan langkah-langkah berikut:

- Buat contoh fungsi Lambda
- Gunakan konsol Lambda untuk melihat AWS SAM template untuk fungsi
- Ekspor konfigurasi fungsi Anda ke AWS Infrastructure Composer dan rancang aplikasi tanpa server sederhana berdasarkan konfigurasi fungsi Anda
- Menyimpan AWS SAM template yang diperbarui yang dapat Anda gunakan sebagai dasar untuk menyebarkan aplikasi tanpa server Anda

Prasyarat

Dalam tutorial ini, Anda menggunakan fitur sinkronisasi lokal Infrastructure Composer untuk menyimpan file template dan kode Anda ke mesin build lokal Anda. Untuk menggunakan fitur ini, Anda memerlukan browser yang mendukung File System Access API, yang memungkinkan aplikasi web membaca, menulis, dan menyimpan file di sistem file lokal Anda. Sebaiknya gunakan Google Chrome atau Microsoft Edge. Untuk informasi selengkapnya tentang API Akses Sistem File, lihat Apa itu API Akses Sistem File?

Buat Fungsi Acara

Pada langkah pertama ini, Anda membuat fungsi Lambda yang dapat Anda gunakan untuk menyelesaikan sisa tutorial. Untuk menjaga hal-hal sederhana, Anda menggunakan konsol Lambda untuk membuat fungsi dasar 'Hello world' menggunakan runtime Python 3.11.

Untuk membuat Fungsi Acara 'Hello world' menggunakan konsol

1. Buka Konsol Lambda.
2. Pilih Buat Fungsi Acara.

3. Biarkan Penulis dari awal dipilih, dan di bawah Informasi dasar, masukkan **`LambdaIaCDemo`** untuk nama Fungsi.
4. Untuk Runtime, pilih Python 3.11.
5. Pilih Buat Fungsi Acara.

Lihat AWS SAM template untuk fungsi Anda

Sebelum Anda mengekspor konfigurasi fungsi Anda ke Infrastructure Composer, gunakan konsol Lambda untuk melihat konfigurasi fungsi Anda saat ini sebagai AWS SAM templat. Dengan mengikuti langkah-langkah di bagian ini, Anda akan belajar tentang anatomi AWS SAM template dan cara mendefinisikan sumber daya seperti fungsi Lambda untuk mulai menentukan aplikasi tanpa server.

Untuk melihat AWS SAM template untuk fungsi Anda

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih fungsi yang baru saja Anda buat (`LambdaIaCDemo`).
3. Di panel Ikhtisar fungsi, pilih Template.

   Di tempat diagram yang mewakili konfigurasi fungsi Anda, Anda akan melihat AWS SAM template untuk fungsi Anda. Template akan terlihat seperti berikut ini.

```
# This AWS SAM template has been generated from your function's
# configuration. If your function has one or more triggers, note
# that the AWS resources associated with these triggers aren't fully
# specified in this template and include placeholder values.Open this template
# in AWS Application Composer or your favorite IDE and modify
# it to specify a serverless application with other AWS resources.
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: An AWS Serverless Specification template describing your function.
Resources:
  LambdaIaCDemo:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: .
      Description: ''
      MemorySize: 128
      Timeout: 3
      Handler: lambda_function.lambda_handler
      Runtime: python3.11
```

```
      Architectures:
        - x86_64
      EventInvokeConfig:
        MaximumEventAgeInSeconds: 21600
        MaximumRetryAttempts: 2
      EphemeralStorage:
        Size: 512
      RuntimeManagementConfig:
        UpdateRuntimeOn: Auto
      SnapStart:
        ApplyOn: None
      PackageType: Zip
      Policies:
        Statement:
          - Effect: Allow
            Action:
              - logs:CreateLogGroup
            Resource: arn:aws:logs:us-east-1:123456789012:*
          - Effect: Allow
            Action:
              - logs:CreateLogStream
              - logs:PutLogEvents
            Resource:
              - >-
                arn:aws:logs:us-east-1:123456789012:log-group:/aws/lambda/
LambdaIaCDemo:*
```

Mari luangkan waktu sejenak untuk melihat template YAMB untuk fungsi Anda dan memahami beberapa konsep kunci.

Template dimulai dengan deklarasi`Transform: AWS::Serverless-2016-10-31`. Deklarasi ini diperlukan karena di balik layar, AWS SAM template digunakan melalui CloudFormation. Menggunakan `Transform` pernyataan mengidentifikasi template sebagai file AWS SAM template.

Setelah `Transform` deklarasi datang `Resources` bagian. Di sinilah AWS sumber daya yang ingin Anda terapkan dengan AWS SAM template Anda ditentukan. AWS SAM template dapat berisi kombinasi AWS SAM sumber daya dan sumber CloudFormation daya. Ini karena selama penerapan, AWS SAM templat diperluas ke CloudFormation templat, sehingga CloudFormation sintaks apa pun yang valid dapat ditambahkan ke templat. AWS SAM

Saat ini, hanya ada satu sumber daya yang ditentukan di `Resources` bagian template, fungsi Lambda Anda. `LambdaIaCDemo` Untuk menambahkan fungsi Lambda ke AWS SAM template, Anda menggunakan jenis `AWS::Serverless::Function` sumber daya. Sumber daya fungsi Lambda menentukan runtime fungsi, penangan fungsi, dan opsi konfigurasi lainnya. `Properties` Jalur ke kode sumber fungsi Anda yang AWS SAM harus digunakan untuk menyebarkan fungsi juga didefinisikan di sini. Untuk mempelajari lebih lanjut tentang sumber daya fungsi Lambda di AWS SAM, lihat AWS::Serverless: :Fungsi dalam Panduan Pengembang.AWS SAM

Selain properti dan konfigurasi fungsi, template juga menentukan kebijakan AWS Identity and Access Management (IAM) untuk fungsi Anda. Kebijakan ini memberikan izin fungsi Anda untuk menulis log ke Amazon CloudWatch Logs. Saat Anda membuat fungsi di konsol Lambda, Lambda secara otomatis melampirkan kebijakan ini ke fungsi Anda. Untuk mempelajari selengkapnya tentang menentukan kebijakan IAM untuk fungsi dalam AWS SAM templat, lihat *`policies`* properti di halaman AWS::Serverless: :Fungsi dari Panduan Pengembang AWS SAM .

Untuk mempelajari lebih lanjut tentang struktur AWS SAM templat, lihat anatomi AWS SAM templat.

Gunakan AWS Infrastructure Composer untuk merancang aplikasi tanpa server

Untuk mulai membangun aplikasi tanpa server sederhana menggunakan AWS SAM template fungsi Anda sebagai titik awal, Anda mengekspor konfigurasi fungsi Anda ke Infrastructure Composer dan mengaktifkan mode sinkronisasi lokal Infrastructure Composer. Sinkronisasi lokal secara otomatis menyimpan kode fungsi dan AWS SAM template Anda ke mesin build lokal Anda dan membuat template tersimpan Anda tetap disinkronkan saat Anda menambahkan AWS sumber daya lain di Infrastructure Composer.

Untuk mengekspor fungsi Anda ke Infrastructure Composer

1. Di panel Ikhtisar Fungsi, pilih Ekspor ke Komposer Aplikasi.

   Untuk mengekspor konfigurasi dan kode fungsi Anda ke Infrastructure Composer, Lambda membuat bucket Amazon S3 di akun Anda untuk menyimpan sementara data ini.
2. Di kotak dialog, pilih Konfirmasi dan buat proyek untuk menerima nama default untuk bucket ini dan ekspor konfigurasi dan kode fungsi Anda ke Infrastructure Composer.
3. (Opsional) Untuk memilih nama lain untuk bucket Amazon S3 yang dibuat Lambda, masukkan nama baru dan pilih Konfirmasi dan buat proyek. Nama bucket Amazon S3 harus unik secara global dan mengikuti aturan penamaan bucket.

Memilih Konfirmasi dan buat proyek membuka konsol Infrastructure Composer. Di kanvas, Anda akan melihat fungsi Lambda Anda.

4. Dari menu tarik-turun, pilih Aktifkan sinkronisasi lokal.
5. Di kotak dialog yang terbuka, pilih Pilih folder dan pilih folder di mesin build lokal Anda.
6. Pilih Aktifkan untuk mengaktifkan sinkronisasi lokal.

Untuk mengekspor fungsi Anda ke Infrastructure Composer, Anda memerlukan izin untuk menggunakan tindakan API tertentu. Jika Anda tidak dapat mengekspor fungsi Anda, lihat the section called “Izin yang diperlukan” dan pastikan Anda memiliki izin yang Anda butuhkan.

> **Note**
>
> Harga Amazon S3 standar berlaku untuk bucket yang dibuat Lambda saat Anda mengekspor fungsi ke Infrastructure Composer. Objek yang dimasukkan Lambda ke dalam bucket secara otomatis dihapus setelah 10 hari, tetapi Lambda tidak menghapus bucket itu sendiri. Untuk menghindari biaya tambahan ditambahkan ke Anda Akun AWS, ikuti petunjuk di Menghapus bucket setelah Anda mengekspor fungsi Anda ke Infrastructure Composer. Untuk informasi selengkapnya tentang bucket Amazon S3 yang dibuat Lambda, lihat. the section called “Komposer Infrastruktur”

Untuk merancang aplikasi tanpa server Anda di Infrastructure Composer

Setelah mengaktifkan sinkronisasi lokal, perubahan yang Anda buat di Infrastructure Composer tercermin dalam AWS SAM template yang disimpan di mesin build lokal Anda. Sekarang Anda dapat menarik dan melepas AWS sumber daya tambahan ke kanvas Infrastructure Composer untuk membangun aplikasi Anda. Dalam contoh ini, Anda menambahkan antrean sederhana Amazon SQS sebagai pemicu fungsi Lambda dan tabel DynamoDB untuk fungsi yang akan menulis data.

1. Tambahkan pemicu Amazon SQS ke fungsi Lambda Anda dengan melakukan hal berikut:
   a. Di bidang pencarian di palet Sumber Daya, masukkan**`SQS`**.
   b. Seret sumber daya SQS Queue ke kanvas Anda dan posisikan di sebelah kiri fungsi Lambda Anda.
   c. Pilih Detail, dan untuk Logical ID masukkan**`LambdaIaCQueue`**.
   d. Pilih Simpan.

e. Hubungkan sumber daya Amazon SQS dan Lambda Anda dengan mengklik port Langganan pada kartu antrian SQS dan menyeretnya ke port sebelah kiri pada kartu fungsi Lambda. Munculnya garis antara dua sumber daya menunjukkan koneksi yang sukses. Infrastructure Composer juga menampilkan pesan di bagian bawah kanvas yang menunjukkan bahwa kedua sumber daya berhasil terhubung.

2. Tambahkan tabel Amazon DynamoDB untuk fungsi Lambda Anda untuk menulis data dengan melakukan hal berikut:

   a. Di bidang pencarian di palet Sumber Daya, masukkan**`DynamoDB`**.

   b. Seret sumber daya DynamoDB Table ke kanvas Anda dan posisikan di sebelah kanan fungsi Lambda Anda.

   c. Pilih Detail, dan untuk Logical ID masukkan**`LambdaIaCTable`**.

   d. Pilih Simpan.

   e. Hubungkan tabel DynamoDB ke fungsi Lambda Anda dengan mengklik port kanan kartu fungsi Lambda dan menyeretnya ke port kiri pada kartu DynamoDB.

Sekarang setelah Anda menambahkan sumber daya tambahan ini, mari kita lihat AWS SAM template yang diperbarui Infrastructure Composer telah dibuat.

Untuk melihat pembaruan Anda AWS SAM templat

- Pada kanvas Infrastructure Composer, pilih Template untuk beralih dari tampilan kanvas ke tampilan template.

AWS SAM Template Anda sekarang harus berisi sumber daya dan properti tambahan berikut:

- Antrian Amazon SQS dengan pengenal `LambdaIaCQueue`

```
LambdaIaCQueue:
    Type: AWS::SQS::Queue
    Properties:
      MessageRetentionPeriod: 345600
```

Saat Anda menambahkan antrean Amazon SQS menggunakan Infrastructure Composer, Infrastructure Composer menetapkan properti. `MessageRetentionPeriod` Anda juga dapat

mengatur **`FifoQueue`** properti dengan memilih Detail pada kartu SQS Queue dan memeriksa atau menghapus centang antrian Fifo.

Untuk mengatur properti lain untuk antrian Anda, Anda dapat mengedit template secara manual untuk menambahkannya. Untuk mempelajari lebih lanjut tentang *`AWS::SQS::Queue`* sumber daya dan properti yang tersedia, lihat AWS: :SQS: :Queue di Panduan Pengguna.CloudFormation

- `Events`Properti dalam definisi fungsi Lambda Anda yang menentukan antrean Amazon SQS sebagai pemicu fungsi

```
Events:
  LambdaIaCQueue:
    Type: SQS
    Properties:
      Queue: !GetAtt LambdaIaCQueue.Arn
      BatchSize: 1
```

`Events`Properti terdiri dari jenis acara dan satu set properti yang bergantung pada jenisnya. Untuk mempelajari tentang perbedaan yang dapat Layanan AWS Anda konfigurasikan untuk memicu fungsi Lambda dan properti yang dapat Anda atur, lihat EventSourcedi Panduan AWS SAM Pengembang.

- Sebuah tabel DynamoDB dengan identifier `LambdaIaCTable`

```
LambdaIaCTable:
    Type: AWS::DynamoDB::Table
    Properties:
      AttributeDefinitions:
        - AttributeName: id
          AttributeType: S
      BillingMode: PAY_PER_REQUEST
      KeySchema:
        - AttributeName: id
          KeyType: HASH
      StreamSpecification:
        StreamViewType: NEW_AND_OLD_IMAGES
```

Ketika Anda menambahkan tabel DynamoDB menggunakan Infrastructure Composer, Anda dapat mengatur kunci tabel Anda dengan memilih Detail pada kartu tabel DynamoDB dan mengedit nilai-nilai kunci. Infrastructure Composer juga menetapkan nilai default untuk sejumlah properti lain termasuk `BillingMode` dan`StreamViewType`.

Untuk mempelajari lebih lanjut tentang properti ini dan properti lain yang dapat Anda tambahkan ke AWS SAM template Anda, lihat AWS: :DynamoDB: :Table di Panduan Pengguna.CloudFormation

- Kebijakan IAM baru yang memberikan izin fungsi Anda untuk melakukan operasi CRUD pada tabel DynamoDB yang Anda tambahkan.

```
Policies:
...
  - DynamoDBCrudPolicy:
    TableName: !Ref LambdaIaCTable
```

AWS SAM Template akhir yang lengkap akan terlihat seperti berikut ini.

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: An AWS Serverless Specification template describing your function.
Resources:
  LambdaIaCDemo:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: .
      Description: ''
      MemorySize: 128
      Timeout: 3
      Handler: lambda_function.lambda_handler
      Runtime: python3.11
      Architectures:
        - x86_64
      EventInvokeConfig:
        MaximumEventAgeInSeconds: 21600
        MaximumRetryAttempts: 2
      EphemeralStorage:
        Size: 512
      RuntimeManagementConfig:
        UpdateRuntimeOn: Auto
      SnapStart:
        ApplyOn: None
      PackageType: Zip
      Policies:
        - Statement:
            - Effect: Allow
```

```
              Action:
                - logs:CreateLogGroup
              Resource: arn:aws:logs:us-east-1:594035263019:*
            - Effect: Allow
              Action:
                - logs:CreateLogStream
                - logs:PutLogEvents
              Resource:
                - arn:aws:logs:us-east-1:594035263019:log-group:/aws/lambda/
LambdaIaCDemo:*
        - DynamoDBCrudPolicy:
            TableName: !Ref LambdaIaCTable
      Events:
        LambdaIaCQueue:
          Type: SQS
          Properties:
            Queue: !GetAtt LambdaIaCQueue.Arn
            BatchSize: 1
      Environment:
        Variables:
          LAMBDAIACTABLE_TABLE_NAME: !Ref LambdaIaCTable
          LAMBDAIACTABLE_TABLE_ARN: !GetAtt LambdaIaCTable.Arn
  LambdaIaCQueue:
    Type: AWS::SQS::Queue
    Properties:
      MessageRetentionPeriod: 345600
  LambdaIaCTable:
    Type: AWS::DynamoDB::Table
    Properties:
      AttributeDefinitions:
        - AttributeName: id
          AttributeType: S
      BillingMode: PAY_PER_REQUEST
      KeySchema:
        - AttributeName: id
          KeyType: HASH
      StreamSpecification:
        StreamViewType: NEW_AND_OLD_IMAGES
```

Menerapkan aplikasi tanpa server Anda menggunakan AWS SAM (opsional)

Jika Anda ingin menggunakan AWS SAM untuk menyebarkan aplikasi tanpa server menggunakan template yang baru saja Anda buat di Infrastructure Composer, Anda harus menginstal file terlebih dahulu. AWS SAM CLI Untuk melakukan ini, ikuti instruksi di Instalasi AWS SAM CLI.

Sebelum Anda menerapkan aplikasi Anda, Anda juga perlu memperbarui kode fungsi yang disimpan Infrastructure Composer bersama dengan template Anda. Saat ini, `lambda_function.py` file yang disimpan oleh Infrastructure Composer hanya berisi kode dasar 'Hello world' yang disediakan Lambda saat Anda membuat fungsi.

Untuk memperbarui kode fungsi Anda, salin kode berikut dan tempel ke `lambda_function.py` file Infrastructure Composer yang disimpan ke mesin build lokal Anda. Anda menentukan direktori untuk Infrastructure Composer untuk menyimpan file ini ketika Anda mengaktifkan mode Local Sync.

Kode ini menerima pasangan nilai kunci dalam pesan dari antrian Amazon SQS yang Anda buat di Infrastructure Composer. Jika kedua kunci dan nilai adalah string, kode kemudian menggunakannya untuk menulis item ke tabel DynamoDB didefinisikan dalam template Anda.

Kode fungsi Python yang diperbarui

```
import boto3
import os
import json

# define the DynamoDB table that Lambda will connect to
tablename = os.environ['LAMBDAIACTABLE_TABLE_NAME']

# create the DynamoDB resource
dynamo = boto3.client('dynamodb')

def lambda_handler(event, context):
    # get the message out of the SQS event
    message = event['Records'][0]['body']
    data = json.loads(message)
    # write event data to DDB table
    if check_message_format(data):
        key = next(iter(data))
        value = data[key]
        dynamo.put_item(
            TableName=tablename,
            Item={
```

```
                'id': {'S': key},
                'Value': {'S': value}
            }
        )
    else:
        raise ValueError("Input data not in the correct format")

# check that the event object contains a single key value
# pair that can be written to the database
def check_message_format(message):
    if len(message) != 1:
        return False

    key, value = next(iter(message.items()))

    if not (isinstance(key, str) and isinstance(value, str)):
        return False

    else:
        return True
```

Untuk menerapkan aplikasi tanpa server

Untuk menyebarkan aplikasi Anda menggunakan AWS SAM CLI, lakukan langkah-langkah berikut. Agar fungsi Anda dapat membangun dan menerapkan dengan benar, Python versi 3.11 harus diinstal pada mesin build Anda dan di mesin Anda. `PATH`

1. Jalankan perintah berikut dari direktori di mana Infrastructure Composer menyimpan `lambda_function.py` file Anda `template.yaml` dan.

   ```
   sam build
   ```

   Perintah ini mengumpulkan artefak build untuk aplikasi Anda dan menempatkannya dalam format dan lokasi yang tepat untuk menerapkannya.

2. Untuk menerapkan aplikasi Anda dan membuat sumber daya Lambda, Amazon SQS, dan DynamoDB yang ditentukan AWS SAM dalam template Anda, jalankan perintah berikut.

   ```
   sam deploy --guided
   ```

Menggunakan `--guided` tanda berarti itu AWS SAM akan menunjukkan kepada Anda petunjuk untuk memandu Anda melalui proses penerapan. Untuk penerapan ini, terima opsi default dengan menekan Enter.

Selama proses penerapan, AWS SAM buat sumber daya berikut di: Akun AWS

- Sebuah CloudFormation tumpukan bernama `sam-app`
- Fungsi Lambda dengan format nama `sam-app-LambdaIaCDemo-`*`99VXPpYQVv1M`*
- Antrian Amazon SQS dengan format nama `sam-app-LambdaIaCQueue-`*`xL87VeKsGiIo`*
- Sebuah tabel DynamoDB dengan format nama `sam-app-LambdaIaCTable-`*`CN0S66C0VLNV`*

AWS SAM juga membuat peran dan kebijakan IAM yang diperlukan sehingga fungsi Lambda Anda dapat membaca pesan dari antrian Amazon SQS dan melakukan operasi CRUD pada tabel DynamoDB.

Menguji aplikasi yang Anda gunakan (opsional)

Untuk mengonfirmasi bahwa aplikasi tanpa server Anda diterapkan dengan benar, kirim pesan ke antrian Amazon SQS Anda yang berisi pasangan nilai kunci dan periksa apakah Lambda menulis item ke tabel DynamoDB Anda menggunakan nilai-nilai ini.

Untuk menguji aplikasi tanpa server Anda

1. Buka halaman Antrian konsol Amazon SQS dan pilih antrian AWS SAM yang dibuat dari template Anda. Namanya memiliki format`sam-app-LambdaIaCQueue-`*`xL87VeKsGiIo`*.
2. Pilih Kirim dan terima pesan dan tempelkan JSON berikut ke Konten pesan di bagian Kirim pesan.

   ```
   {
       "myKey": "myValue"
   }
   ```

3. Pilih Kirim pesan.

   Mengirim pesan ke antrian menyebabkan Lambda memanggil fungsi Anda melalui pemetaan sumber peristiwa yang ditentukan dalam templat Anda. AWS SAM Untuk mengonfirmasi bahwa

Lambda telah memanggil fungsi Anda seperti yang diharapkan, konfirmasikan bahwa item telah ditambahkan ke tabel DynamoDB Anda.

4. Buka halaman Tabel konsol DynamoDB dan pilih tabel Anda. Namanya memiliki formatsam-app-LambdaIaCTable-*CN0S66C0VLNV*.
5. Pilih Jelajahi item tabel. Di panel Item yang dikembalikan, Anda akan melihat item dengan id myKey dan NilaimyValue.

Menerapkan Fungsi Acara dengan AWS CDK

AWS Cloud Development Kit (AWS CDK) Ini adalah kerangka infrastruktur sebagai kode (IAc) yang dapat Anda gunakan untuk mendefinisikan infrastruktur AWS cloud dengan menggunakan bahasa pemrograman pilihan Anda. Untuk menentukan infrastruktur cloud Anda sendiri, pertama-tama Anda menulis aplikasi (dalam salah satu bahasa yang didukung CDK) yang berisi satu atau beberapa tumpukan. Kemudian, Anda mensintesisnya ke CloudFormation template dan menyebarkan sumber daya Anda ke template Anda. Akun AWS Ikuti langkah-langkah dalam topik ini untuk menerapkan Fungsi Peristiwa Lambda yang menampilkan peristiwa dari titik akhir Amazon API Gateway.

Perpustakaan AWS Konstruksi, disertakan dengan CDK, menyediakan modul yang dapat Anda gunakan untuk memodelkan sumber daya yang Layanan AWS disediakan. Untuk layanan populer, perpustakaan menyediakan konstruksi yang dikuratori dengan default cerdas dan praktik terbaik. Anda dapat menggunakan modul aws_lambda untuk menentukan fungsi Anda dan sumber daya pendukung hanya dengan beberapa baris kode.

Prasyarat

Sebelum memulai tutorial ini, instal AWS CDK dengan menjalankan perintah berikut.

```
npm install -g aws-cdk
```

Langkah 1: Siapkan AWS CDK proyek

Buat direktori untuk AWS CDK aplikasi baru Anda dan inisialisasi proyek.

JavaScript

```
mkdir hello-lambda
cd hello-lambda
cdk init --language javascript
```

TypeScript

```
mkdir hello-lambda
cd hello-lambda
cdk init --language typescript
```

Python

```
mkdir hello-lambda
cd hello-lambda
cdk init --language python
```

Setelah proyek dimulai, aktifkan lingkungan virtual proyek dan instal dependensi dasar untuk. AWS CDK

```
source .venv/bin/activate
python -m pip install -r requirements.txt
```

Java

```
mkdir hello-lambda
cd hello-lambda
cdk init --language java
```

Impor proyek Maven ini ke lingkungan pengembangan terintegrasi Java (IDE) Anda. Misalnya, di Eclipse, pilih File, Import, Maven, Existing Maven Projects.

C#

```
mkdir hello-lambda
cd hello-lambda
cdk init --language csharp
```

> **Note**
>
> Template AWS CDK aplikasi menggunakan nama direktori proyek untuk menghasilkan nama untuk file sumber dan kelas. Dalam contoh ini, direktori diberi nama`hello-lambda`. Jika Anda menggunakan nama direktori project yang berbeda, aplikasi Anda tidak akan cocok dengan petunjuk ini.

AWS CDK v2 menyertakan konstruksi stabil untuk semua Layanan AWS dalam satu paket yang dipanggil`aws-cdk-lib`. Paket ini diinstal sebagai dependensi ketika Anda menginisialisasi proyek. Saat bekerja dengan bahasa pemrograman tertentu, paket diinstal saat Anda membangun proyek untuk pertama kalinya.

Langkah 2: Tentukan AWS CDK tumpukan

Tumpukan CDK adalah kumpulan dari satu atau lebih konstruksi, yang mendefinisikan AWS sumber daya. Setiap tumpukan CDK mewakili CloudFormation tumpukan di aplikasi CDK Anda.

Untuk menentukan tumpukan CDK Anda, ikuti instruksi untuk bahasa pemrograman pilihan Anda. Tumpukan ini mendefinisikan yang berikut:

- Nama logis fungsi: `MyFunction`
- Lokasi kode fungsi, yang ditentukan dalam `code` properti. Untuk informasi selengkapnya, lihat Kode handler di Referensi AWS Cloud Development Kit (AWS CDK) API.
- Nama logis REST API: `HelloApi`
- Nama logis titik akhir API Gateway: `ApiGwEndpoint`

Perhatikan bahwa semua tumpukan CDK dalam tutorial ini menggunakan Node.js runtime untuk fungsi Lambda. Anda dapat menggunakan bahasa pemrograman yang berbeda untuk tumpukan CDK dan fungsi Lambda untuk menggunakan kekuatan masing-masing bahasa. Misalnya, Anda dapat menggunakan TypeScript tumpukan CDK untuk menggunakan manfaat pengetikan statis untuk kode infrastruktur Anda. Anda dapat menggunakan JavaScript fungsi Lambda untuk memanfaatkan fleksibilitas dan perkembangan cepat dari bahasa yang diketik secara dinamis.

JavaScript

Buka `lib/hello-lambda-stack.js` file dan ganti isinya dengan yang berikut ini.

```
const { Stack } = require('aws-cdk-lib');
const lambda = require('aws-cdk-lib/aws-lambda');
const apigw = require('aws-cdk-lib/aws-apigateway');

class HelloLambdaStack extends Stack {
  /**
   *
   * @param {Construct} scope
   * @param {string} id
   * @param {StackProps=} props
```

```
   */
  constructor(scope, id, props) {
    super(scope, id, props);
    const fn = new lambda.Function(this, 'MyFunction', {
      code: lambda.Code.fromAsset('lib/lambda-handler'),
      runtime: lambda.Runtime.NODEJS_LATEST,
      handler: 'index.handler'
    });

    const endpoint = new apigw.LambdaRestApi(this, 'MyEndpoint', {
      handler: fn,
      restApiName: "HelloApi"
    });

  }
}

module.exports = { HelloLambdaStack }
```

TypeScript

Buka `lib/hello-lambda-stack.ts` file dan ganti isinya dengan yang berikut ini.

```
import * as cdk from 'aws-cdk-lib';
import { Construct } from 'constructs';
import * as apigw from "aws-cdk-lib/aws-apigateway";
import * as lambda from "aws-cdk-lib/aws-lambda";
import * as path from 'node:path';

export class HelloLambdaStack extends cdk.Stack {
  constructor(scope: Construct, id: string, props?: cdk.StackProps){
    super(scope, id, props)
    const fn = new lambda.Function(this, 'MyFunction', {
      runtime: lambda.Runtime.NODEJS_LATEST,
      handler: 'index.handler',
      code: lambda.Code.fromAsset(path.join(__dirname, 'lambda-handler')),
    });

    const endpoint = new apigw.LambdaRestApi(this, `ApiGwEndpoint`, {
      handler: fn,
      restApiName: `HelloApi`,
    });

  }
```

```
}
```

Python

Buka `/hello-lambda/hello_lambda/hello_lambda_stack.py` file dan ganti isinya dengan yang berikut ini.

```
from aws_cdk import (
    Stack,
    aws_apigateway as apigw,
    aws_lambda as _lambda
)
from constructs import Construct

class HelloLambdaStack(Stack):

    def __init__(self, scope: Construct, construct_id: str, **kwargs) -> None:
        super().__init__(scope, construct_id, **kwargs)

        fn = _lambda.Function(
            self,
            "MyFunction",
            runtime=_lambda.Runtime.NODEJS_LATEST,
            handler="index.handler",
            code=_lambda.Code.from_asset("lib/lambda-handler")
        )

        endpoint = apigw.LambdaRestApi(
            self,
            "ApiGwEndpoint",
            handler=fn,
            rest_api_name="HelloApi"
        )
```

Java

Buka `/hello-lambda/src/main/java/com/myorg/HelloLambdaStack.java` file dan ganti isinya dengan yang berikut ini.

```
package com.myorg;

import software.constructs.Construct;
import software.amazon.awscdk.Stack;
```

```
import software.amazon.awscdk.StackProps;
import software.amazon.awscdk.services.apigateway.LambdaRestApi;
import software.amazon.awscdk.services.lambda.Function;

public class HelloLambdaStack extends Stack {
    public HelloLambdaStack(final Construct scope, final String id) {
        this(scope, id, null);
    }

    public HelloLambdaStack(final Construct scope, final String id, final StackProps
 props) {
        super(scope, id, props);

        Function hello = Function.Builder.create(this, "MyFunction")

 .runtime(software.amazon.awscdk.services.lambda.Runtime.NODEJS_LATEST)

 .code(software.amazon.awscdk.services.lambda.Code.fromAsset("lib/lambda-handler"))
                            .handler("index.handler")
                            .build();

        LambdaRestApi api = LambdaRestApi.Builder.create(this, "ApiGwEndpoint")
                                .restApiName("HelloApi")
                                .handler(hello)
                                .build();

    }
}
```

C#

Buka `src/HelloLambda/HelloLambdaStack.cs` file dan ganti isinya dengan yang berikut ini.

```
using Amazon.CDK;
using Amazon.CDK.AWS.APIGateway;
using Amazon.CDK.AWS.Lambda;
using Constructs;

namespace HelloLambda
{
    public class HelloLambdaStack : Stack
    {
        internal HelloLambdaStack(Construct scope, string id, IStackProps props =
 null) : base(scope, id, props)
```

```
        {
            var fn = new Function(this, "MyFunction", new FunctionProps
            {
                Runtime = Runtime.NODEJS_LATEST,
                Code = Code.FromAsset("lib/lambda-handler"),
                Handler = "index.handler"
            });

            var api = new LambdaRestApi(this, "ApiGwEndpoint", new
 LambdaRestApiProps
            {
                Handler = fn
            });
        }
    }
}
```

Langkah 3: Buat kode Fungsi Acara

1. Dari root project Anda (`hello-lambda`), buat `/lib/lambda-handler` direktori untuk kode fungsi Lambda. Direktori ini ditentukan dalam `code` properti AWS CDK tumpukan Anda.
2. Buat file baru yang disebut `index.js` di `/lib/lambda-handler` direktori. Tempel kode berikut ke file. Fungsi mengekstrak properti tertentu dari permintaan API dan mengembalikannya sebagai respons JSON.

```
exports.handler = async (event) => {
  // Extract specific properties from the event object
  const { resource, path, httpMethod, headers, queryStringParameters, body } =
 event;
  const response = {
    resource,
    path,
    httpMethod,
    headers,
    queryStringParameters,
    body,
  };
  return {
    body: JSON.stringify(response, null, 2),
    statusCode: 200,
  };
```

```
};
```

Langkah 4: Menyebarkan AWS CDK tumpukan

1. Dari root proyek Anda, jalankan perintah cdk synth:

```
cdk synth
```

Perintah ini mensintesis AWS CloudFormation template dari tumpukan CDK Anda. Template adalah file YAMM sekitar 400 baris, mirip dengan yang berikut ini.

> **Note**
>
> Jika Anda mendapatkan kesalahan berikut, pastikan Anda berada di root direktori proyek Anda.
>
> ```
> --app is required either in command-line, in cdk.json or in ~/.cdk.json
> ```

Example CloudFormation templat

```
Resources:
  MyFunctionServiceRole3C357FF2:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Statement:
          - Action: sts:AssumeRole
            Effect: Allow
            Principal:
              Service: lambda.amazonaws.com
        Version: "2012-10-17"
      ManagedPolicyArns:
        - Fn::Join:
            - ""
            - - "arn:"
              - Ref: AWS::Partition
              - :iam::aws:policy/service-role/AWSLambdaBasicExecutionRole
    Metadata:
```

```
      aws:cdk:path: HelloLambdaStack/MyFunction/ServiceRole/Resource
  MyFunction1BAA52E7:
    Type: AWS::Lambda::Function
    Properties:
      Code:
        S3Bucket:
          Fn::Sub: cdk-hnb659fds-assets-${AWS::AccountId}-${AWS::Region}
        S3Key:
ab1111111cd32708dc4b83e81a21c296d607ff2cdef00f1d7f48338782f92l3901.zip
      Handler: index.handler
      Role:
        Fn::GetAtt:
          - MyFunctionServiceRole3C357FF2
          - Arn
      Runtime: nodejs24.x
      ...
```

2. Jalankan perintah cdk deploy:

```
cdk deploy
```

Tunggu sementara sumber daya Anda dibuat. Output akhir menyertakan URL untuk titik akhir API Gateway Anda. Contoh:

```
Outputs:
HelloLambdaStack.ApiGwEndpoint77F417B1 = https://abcd1234.execute-api.us-
east-1.amazonaws.com/prod/
```

Langkah 5: Uji fungsinya

Untuk menjalankan fungsi Lambda, salin titik akhir API Gateway dan tempelkan ke browser web atau jalankan perintah: `curl`

```
curl -s https://abcd1234.execute-api.us-east-1.amazonaws.com/prod/
```

Responsnya adalah representasi JSON dari properti yang dipilih dari objek peristiwa asli, yang berisi informasi tentang permintaan yang dibuat ke titik akhir API Gateway. Contoh:

```
{
  "resource": "/",
  "path": "/",
```

```
  "httpMethod": "GET",
  "headers": {
    "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/
webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7",
    "Accept-Encoding": "gzip, deflate, br, zstd",
    "Accept-Language": "en-US,en;q=0.9",
    "CloudFront-Forwarded-Proto": "https",
    "CloudFront-Is-Desktop-Viewer": "true",
    "CloudFront-Is-Mobile-Viewer": "false",
    "CloudFront-Is-SmartTV-Viewer": "false",
    "CloudFront-Is-Tablet-Viewer": "false",
    "CloudFront-Viewer-ASN": "16509",
    "CloudFront-Viewer-Country": "US",
    "Host": "abcd1234.execute-api.us-east-1.amazonaws.com",
     ...
```

Langkah 6: Bersihkan sumber daya Anda

Titik akhir API Gateway dapat diakses publik. Untuk mencegah tagihan yang tidak terduga, jalankan perintah cdk destroy untuk menghapus tumpukan dan semua sumber daya terkait.

```
cdk destroy
```

Langkah selanjutnya

Untuk informasi tentang menulis AWS CDK aplikasi dalam bahasa pilihan Anda, lihat berikut ini:

TypeScript

- Bekerja dengan AWS CDK di TypeScript

JavaScript

- Bekerja dengan AWS CDK di JavaScript

Python

- Bekerja dengan AWS CDK in Python

Java

- Bekerja dengan AWS CDK di Jawa

C#

- Bekerja dengan AWS CDK di C #

Go

Bekerja dengan AWS CDK in Go

Powertools untuk AWS Lambda

Powertools untuk AWS Lambda (juga disebut sebagai Powertools AWS for) menyediakan fungsi utilitas, dekorator, dan middleware yang menangani tugas-tugas Lambda umum seperti logging terstruktur, tracing, pengumpulan metrik, dan validasi input. Gunakan Powertools untuk AWS Lambda untuk menerapkan praktik terbaik tanpa server dan mempercepat pengembangan di beberapa Fungsi Acara Lambda. Melakukan hal ini menyederhanakan tugas pengembangan umum dalam fungsi Lambda Anda.

Manfaat utama Powertools untuk AWS

Sementara pengembangan Lambda dimungkinkan tanpa Powertools untuk AWS, menggunakannya menawarkan beberapa keuntungan:

- Built-in observabilitas: Pencatatan terstruktur, penelusuran, dan metrik khusus
- Manajemen rahasia: Pengambilan parameter, penanganan rahasia, dan idempotensi
- Peningkatan Progresif: Pilih utilitas yang paling sesuai dengan kebutuhan Anda
- Pengembangan yang dipercepat: Penguraian acara, validasi, dan pemrosesan batch
- Praktik terbaik: Implementasi AWS Well-Architected pola tanpa server

Mengintegrasikan Powertools dengan AWS

Powertools untuk AWS membantu Anda membangun aplikasi tanpa server yang siap produksi dengan kode kustom yang lebih sedikit. Tersedia dalam Python,, .NET TypeScript/Node.js, dan Java, Powertools untuk AWS dapat disertakan melalui Lambda Layers, atau menggunakan manajer paket bahasa. Setiap implementasi bahasa menyediakan fitur inti seperti logging terstruktur, penelusuran, pengumpulan metrik, dan penanganan acara, sambil mempertahankan idiom alami untuk setiap bahasa pemrograman. Implementasi ini dilengkapi dengan komponen khusus untuk integrasi AWS layanan, mendukung pengambilan parameter, pemrosesan batch, dan penanganan API, bersama dengan praktik terbaik seperti propagasi ID korelasi, penanganan kesalahan, dan pola idempotensi. Bersama-sama, fitur-fitur ini memungkinkan pengembang untuk membangun aplikasi tanpa server yang kuat dan dapat dipelihara sekaligus mengurangi overhead kode khusus.

- Powertools untuk AWS Lambda (Python)
- Powertools untuk AWS TypeScript Lambda ()
- Powertools untuk AWS Lambda (Java)
- Powertools para AWS Lambda (.NET)

Langkah selanjutnya

Untuk mempelajari selengkapnya tentang bekerja dengan Powertools AWS, lihat sumber daya berikut:

- Powertools untuk lokakarya AWS Lambda
- Pola tanpa server yang menggunakan Powertools untuk AWS
- AWS lensa tanpa server yang dirancang dengan baik
- Membangun API Tanpa Server dengan Powertools untuk Lambda AWS

Mengelola alur kerja dan acara Lambda

Saat membuat aplikasi tanpa server dengan Lambda, Anda sering membutuhkan cara untuk mengatur eksekusi fungsi dan menangani peristiwa. AWS menyediakan beberapa pendekatan untuk mengoordinasikan Fungsi Acara Lambda:

- Fungsi Lambda yang tahan lama untuk orkestrasi alur kerja kode pertama dalam Lambda
- AWS Step Functions untuk orkestrasi alur kerja visual di beberapa layanan
- Amazon EventBridge Scheduler dan Amazon EventBridge untuk arsitektur dan penjadwalan yang digerakkan oleh acara

Anda juga dapat mengintegrasikan pendekatan ini bersama-sama. Misalnya, Anda dapat menggunakan EventBridge Scheduler untuk memicu fungsi tahan lama atau alur kerja Step Functions saat peristiwa tertentu terjadi, atau mengonfigurasi alur kerja untuk mempublikasikan peristiwa ke EventBridge Scheduler pada titik eksekusi yang ditentukan. Topik berikut di bagian ini memberikan informasi lebih lanjut tentang bagaimana Anda dapat menggunakan opsi orkestrasi ini.

Code-first orkestrasi dengan fungsi tahan lama

Fungsi tahan lama Lambda menyediakan pendekatan kode-pertama untuk orkestrasi alur kerja, memungkinkan Anda untuk membangun alur kerja stateful dan berjalan lama secara langsung dalam

fungsi Lambda Anda. Tidak seperti layanan orkestrasi eksternal, fungsi tahan lama menjaga logika alur kerja Anda dalam kode, membuatnya lebih mudah untuk versi, pengujian, dan pemeliharaan bersama logika bisnis Anda.

Fungsi tahan lama sangat ideal saat Anda membutuhkan:

- Gunakan bahasa pemrograman standar: Tentukan alur kerja menggunakan bahasa pemrograman yang sudah dikenal seperti JavaScript dan Python
- Long-running proses: Jalankan alur kerja yang dapat berjalan hingga satu tahun, jauh melampaui batas pemanggilan 15 menit default
- Pengembangan yang disederhanakan: Simpan logika alur kerja dan logika bisnis dalam basis kode yang sama untuk pemeliharaan dan pengujian yang lebih mudah
- Cost-effective waiting: Jeda eksekusi selama status tunggu tanpa menghabiskan sumber daya komputasi
- Built-in manajemen status: Checkpointing otomatis dan persistensi status tanpa konfigurasi penyimpanan eksternal

Untuk bantuan memilih antara fungsi tahan lama dan Step Functions, lihat Fungsi tahan lama atau Step Functions.

Untuk informasi selengkapnya tentang fungsi tahan lama, lihat Fungsi tahan lama Lambda.

Mengatur alur kerja dengan Step Functions

AWS Step Functions adalah layanan orkestrasi alur kerja yang membantu Anda mengoordinasikan beberapa fungsi Lambda dan layanan lainnya ke dalam alur kerja terstruktur. AWS Alur kerja ini dapat mempertahankan status, menangani kesalahan dengan mekanisme coba ulang yang canggih, dan memproses data dalam skala besar.

Step Functions menawarkan dua jenis alur kerja untuk memenuhi kebutuhan orkestrasi yang berbeda:

Alur kerja standar

> Ideal untuk alur kerja yang berjalan lama dan dapat diaudit yang memerlukan semantik eksekusi tepat sekali. Alur kerja standar dapat berjalan hingga satu tahun, memberikan riwayat eksekusi terperinci, dan mendukung debugging visual. Mereka cocok untuk proses seperti pemenuhan pesanan, pipa pemrosesan data, atau pekerjaan analitik multi-langkah.

Alur kerja ekspres

Dirancang untuk beban kerja berdurasi pendek dengan kecepatan peristiwa tinggi dengan semantik eksekusi paling sedikit sekali. Alur kerja ekspres dapat berjalan hingga lima menit dan ideal untuk pemrosesan peristiwa volume tinggi, transformasi data streaming, atau skenario konsumsi data IoT. Mereka menawarkan throughput yang lebih tinggi dan biaya yang berpotensi lebih rendah dibandingkan dengan alur kerja Standar.

> **Note**
>
> Untuk informasi selengkapnya tentang jenis alur kerja Step Functions, lihat Memilih jenis alur kerja di Step Functions.

Dalam alur kerja ini, Step Functions menyediakan dua jenis status Peta untuk pemrosesan paralel:

Peta Inline

Memproses item dari array JSON dalam riwayat eksekusi alur kerja induk. Peta Inline mendukung hingga 40 iterasi bersamaan dan cocok untuk kumpulan data yang lebih kecil atau ketika Anda perlu menyimpan semua pemrosesan dalam satu eksekusi. Untuk informasi selengkapnya, lihat Menggunakan status Peta dalam mode Inline.

Peta Terdistribusi

Memungkinkan pemrosesan beban kerja paralel skala besar dengan mengulangi kumpulan data yang melebihi 256 KiB atau memerlukan lebih dari 40 iterasi bersamaan. Dengan dukungan hingga 10.000 eksekusi alur kerja anak paralel, Peta Terdistribusi unggul dalam memproses data semi-terstruktur yang disimpan di Amazon S3, seperti file JSON atau CSV, sehingga ideal untuk pemrosesan batch dan operasi ETL. Untuk informasi selengkapnya, lihat Menggunakan status Peta dalam mode Terdistribusi.

Dengan menggabungkan jenis alur kerja dan status Peta ini, Step Functions menyediakan perangkat yang fleksibel dan kuat untuk mengatur aplikasi tanpa server yang kompleks, mulai dari operasi skala kecil hingga jaringan pemrosesan data skala besar.

Untuk memulai menggunakan Lambda dengan Step Functions, lihat Mengatur fungsi Lambda dengan Step Functions.

Mengelola acara dengan EventBridge dan EventBridge Scheduler

Amazon EventBridge adalah layanan bus acara yang membantu Anda membangun arsitektur berbasis acara. Ini merutekan peristiwa antara AWS layanan, aplikasi terintegrasi, dan aplikasi perangkat lunak sebagai layanan (SaaS). EventBridge Scheduler adalah penjadwal tanpa server. Dengan EventBridge Scheduler, Anda dapat membuat, menjalankan, dan mengelola tugas dari satu layanan pusat, memanggil fungsi Lambda pada jadwal menggunakan ekspresi cron dan rate, atau mengonfigurasi pemanggilan satu kali.

Amazon EventBridge dan EventBridge Scheduler membantu Anda membangun arsitektur berbasis peristiwa dengan Lambda. EventBridge merutekan peristiwa antara AWS layanan, aplikasi terintegrasi, dan aplikasi SaaS, sementara EventBridge Scheduler menyediakan kemampuan penjadwalan khusus untuk menjalankan fungsi Lambda secara berulang atau satu kali.

Layanan ini menyediakan beberapa kemampuan utama untuk bekerja dengan fungsi Lambda:

- Buat aturan yang cocok dan rute acara ke fungsi Lambda menggunakan EventBridge
- Siapkan pemanggilan fungsi berulang menggunakan ekspresi cron dan rate dengan Scheduler EventBridge
- Konfigurasikan pemanggilan fungsi satu kali pada tanggal dan waktu tertentu
- Tentukan jendela waktu yang fleksibel dan coba lagi kebijakan untuk pemanggilan terjadwal

Lihat informasi yang lebih lengkap di Memanggil Fungsi Acara Lambda sesuai jadwal.

Cara menguji fungsi dan aplikasi tanpa server

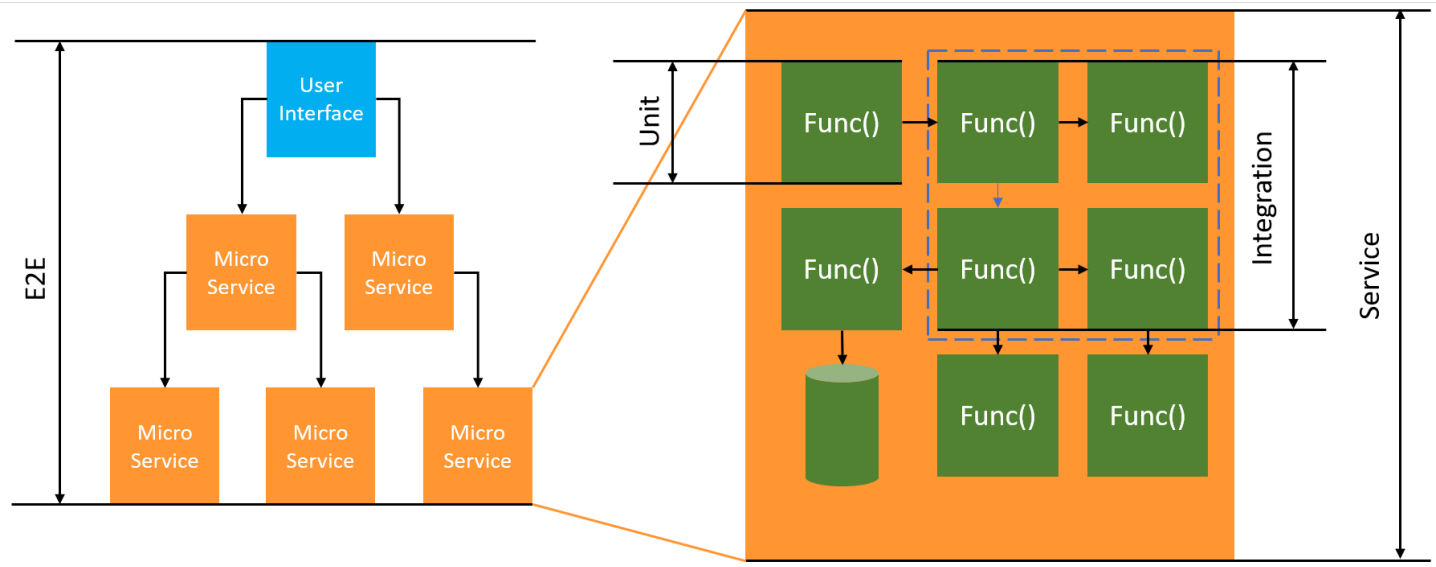
Menguji fungsi tanpa server menggunakan jenis dan teknik pengujian tradisional, tetapi Anda juga harus mempertimbangkan pengujian aplikasi tanpa server secara keseluruhan. Cloud-based tes memberikan ukuran kualitas yang paling akurat dari fungsi dan aplikasi tanpa server Anda.

Arsitektur aplikasi tanpa server mencakup layanan terkelola yang menyediakan fungsionalitas aplikasi penting melalui panggilan API. Untuk alasan ini, siklus pengembangan Anda harus menyertakan pengujian otomatis yang memverifikasi fungsionalitas saat fungsi dan layanan Anda berinteraksi.

Jika Anda tidak membuat pengujian berbasis cloud, Anda dapat mengalami masalah karena perbedaan antara lingkungan lokal dan lingkungan yang diterapkan. Proses integrasi berkelanjutan

Anda harus menjalankan pengujian terhadap serangkaian sumber daya yang disediakan di cloud sebelum mempromosikan kode Anda ke lingkungan penerapan berikutnya, seperti QA, Staging, atau Production.

Lanjutkan membaca panduan singkat ini untuk mempelajari strategi pengujian untuk aplikasi tanpa server, atau kunjungi repositori Sampel Uji Tanpa Server untuk menyelami contoh-contoh praktis, khusus untuk bahasa dan runtime pilihan Anda.



Untuk pengujian tanpa server, Anda masih menulis pengujian unit, integrasi, dan ujung-ke-ujung.

- Tes unit - Tes yang dijalankan terhadap blok kode yang terisolasi. Misalnya, memverifikasi logika bisnis untuk menghitung biaya pengiriman yang diberikan item dan tujuan tertentu.
- Tes integrasi - Tes yang melibatkan dua atau lebih komponen atau layanan yang berinteraksi, biasanya di lingkungan cloud. Misalnya, memverifikasi fungsi memproses peristiwa dari antrian.
- End-to-end tes - Tes yang memverifikasi perilaku di seluruh aplikasi. Misalnya, memastikan infrastruktur diatur dengan benar dan bahwa peristiwa mengalir antar layanan seperti yang diharapkan untuk merekam pesanan pelanggan.

Hasil bisnis yang ditargetkan

Menguji solusi tanpa server mungkin membutuhkan lebih banyak waktu untuk disiapkan. Anda harus memverifikasi interaksi berbasis peristiwa antar layanan. Ingatlah alasan bisnis praktis ini saat Anda membaca panduan ini:

- Tingkatkan kualitas aplikasi Anda

- Kurangi waktu untuk membangun fitur dan memperbaiki bug

Kualitas aplikasi tergantung pada pengujian banyak skenario. Pertimbangkan skenario bisnis Anda dan otomatiskan pengujian untuk dijalankan terhadap layanan cloud. Ini meningkatkan kualitas aplikasi Anda.

Bug dan masalah konfigurasi lebih murah saat ditangkap di awal siklus pengembangan. Masalah yang tidak terdeteksi sampai produksi membutuhkan lebih banyak usaha dan lebih banyak orang untuk memperbaikinya.

Strategi pengujian tanpa server yang baik meningkatkan kualitas perangkat lunak dan mempercepat iterasi. Ini memverifikasi bahwa Fungsi dan aplikasi Peristiwa Lambda Anda berfungsi seperti yang diharapkan di cloud.

Apa yang harus diuji

Kami merekomendasikan strategi pengujian yang menguji perilaku layanan terkelola, konfigurasi cloud, kebijakan keamanan, dan integrasi dengan kode Anda. Pengujian perilaku, juga dikenal sebagai pengujian kotak hitam, memverifikasi bahwa sistem bekerja seperti yang diharapkan tanpa mengetahui internal.

- Jalankan pengujian unit untuk memeriksa logika bisnis di dalam fungsi Lambda.
- Verifikasi layanan terintegrasi benar-benar dipanggil, dan parameter input sudah benar.
- Periksa apakah suatu peristiwa melewati semua layanan yang diharapkan secara end-to-end dalam alur kerja.

Dalam arsitektur berbasis server tradisional, tim sering menguji hanya kode yang berjalan pada server aplikasi. Mereka menganggap komponen, layanan, atau dependensi lain sebagai eksternal dan di luar cakupan.

Aplikasi tanpa server terdiri dari unit kerja kecil. Contohnya termasuk fungsi Lambda yang mengambil produk dari database, memproses item dari antrian, atau mengubah ukuran gambar dalam penyimpanan. Setiap komponen berjalan di lingkungannya sendiri. Tim mengelola banyak unit kecil ini dalam satu aplikasi.

Beberapa fungsi dapat ditangani sepenuhnya oleh layanan terkelola seperti Amazon S3, atau dibangun tanpa kode khusus. Anda tidak perlu menguji layanan terkelola ini. Namun, Anda harus menguji bagaimana kode Anda terintegrasi dengan mereka.

Cara menguji tanpa server

Anda mungkin tahu cara menguji aplikasi yang digunakan secara lokal. Anda menulis tes terhadap kode di desktop Anda atau di dalam wadah. Misalnya, Anda mungkin memanggil layanan web lokal dan kemudian memeriksa responsnya.

Solusi tanpa server menggunakan kode fungsi dan layanan terkelola berbasis cloud, seperti antrian, database, bus acara, dan sistem pesan. Komponen-komponen ini terhubung melalui arsitektur yang digerakkan oleh peristiwa, di mana pesan, yang disebut peristiwa, mengalir dari satu sumber daya ke sumber daya lainnya. Beberapa interaksi sinkron, seperti layanan web yang mengembalikan hasil segera.

Lainnya asinkron, seperti menempatkan item dalam antrian atau memulai langkah alur kerja. Strategi pengujian Anda harus mencakup kedua jenis dan menguji interaksi antar layanan. Untuk interaksi asinkron, Anda mungkin perlu mendeteksi efek samping pada komponen hilir yang tidak segera terlihat.

Anda tidak dapat sepenuhnya mereplikasi lingkungan cloud secara lokal. Ini termasuk antrian, tabel database, bus acara, dan kebijakan keamanan. Perbedaan antara lingkungan lokal dan cloud menyebabkan masalah. Perbedaan ini meningkatkan waktu untuk mereproduksi dan memperbaiki bug.

Dalam aplikasi tanpa server, komponen ada sepenuhnya di cloud. Pengujian terhadap kode dan layanan cloud diperlukan untuk mengembangkan fitur dan memperbaiki bug.

Teknik pengujian

Strategi pengujian Anda mungkin mencakup campuran teknik. Anda menggunakan tes interaktif cepat untuk men-debug fungsi di konsol. Anda menulis pengujian unit otomatis untuk memeriksa logika bisnis. Anda memverifikasi panggilan ke layanan eksternal dengan tiruan. Anda juga dapat menguji emulator yang meniru layanan.

- the section called “Pengujian di cloud”: Anda menerapkan infrastruktur dan kode untuk menguji dengan layanan, kebijakan keamanan, dan konfigurasi yang sebenarnya. Cloud-based tes memberikan ukuran kualitas kode Anda yang paling akurat.

  Mendebug fungsi di konsol adalah cara cepat untuk menguji di cloud. Anda dapat memilih dari contoh peristiwa uji atau membuat acara khusus. Anda juga dapat berbagi acara pengujian dengan tim Anda melalui konsol.

Untuk mengotomatiskan pengujian dalam siklus hidup pengembangan dan pembuatan, uji di luar konsol. Lihat bagian pengujian khusus bahasa dalam panduan ini untuk strategi otomatisasi.

- the section called “Menguji dengan tiruan”: Mocks adalah objek dalam kode Anda yang mensimulasikan layanan eksternal. Mereka memberikan perilaku yang telah ditentukan sebelumnya untuk memverifikasi panggilan dan parameter layanan. Palsu adalah tiruan yang mengambil jalan pintas untuk menyederhanakan atau mempercepat pengujian. Misalnya, objek akses data palsu mungkin mengembalikan data dari datastore dalam memori. Mocks dapat menyederhanakan dependensi yang kompleks, tetapi dapat menyebabkan lebih banyak tiruan untuk menggantikan dependensi bersarang.
- the section called “Menguji secara lokal menggunakan AWS SAM CLI”: Gunakan AWS SAM CLI untuk memanggil fungsi Lambda secara lokal di wadah Docker yang menggunakan lingkungan runtime yang sama dengan Lambda. AWS Anda dapat menguji logika fungsi dan pemrosesan peristiwa tanpa menerapkan ke cloud.
- the section called “Pengujian dengan emulasi”: Gunakan LocalStack integrasi dalam VS Code untuk meniru beberapa AWS layanan secara lokal untuk menguji integrasi layanan.

Pengujian di cloud

Pengujian di cloud sangat berharga untuk semua fase pengujian: pengujian unit, pengujian integrasi, dan pengujian ujung ke ujung. Pengujian yang dijalankan terhadap kode dan layanan berbasis cloud memberikan ukuran kualitas kode yang paling akurat.

Cara sederhana untuk menjalankan fungsi Lambda di cloud adalah dengan acara pengujian di file. Konsol Manajemen AWSPeristiwa pengujian adalah input JSON ke fungsi Anda. Jika fungsi Anda tidak memerlukan input, acara dapat berupa dokumen (`{}`) JSON kosong. Konsol menyediakan contoh peristiwa untuk banyak integrasi layanan. Anda dapat berbagi acara dengan tim Anda untuk mempermudah pengujian.

Pelajari cara men-debug fungsi sampel di konsol.

> **Note**
>
> Meskipun menjalankan fungsi di konsol adalah cara cepat untuk men-debug, mengotomatiskan siklus pengujian Anda sangat penting untuk meningkatkan kualitas aplikasi dan kecepatan pengembangan.

Sampel otomatisasi pengujian tersedia di repositori Sampel Uji Tanpa Server. Baris perintah berikut menjalankan contoh uji integrasi Python otomatis:

```
python -m pytest -s tests/integration -v
```

Meskipun pengujian berjalan secara lokal, ia berbicara dengan sumber daya berbasis cloud. Sumber daya ini digunakan menggunakan alat AWS SAM baris perintah AWS Serverless Application Model dan. Kode pengujian pertama-tama mengambil output tumpukan yang diterapkan, seperti titik akhir API, fungsi ARN, dan peran keamanan.

Kemudian, ia mengirimkan permintaan ke titik akhir API. Tanggapan berisi daftar ember Amazon S3. Pengujian ini dijalankan terhadap sumber daya berbasis cloud untuk memverifikasi bahwa sumber daya tersebut digunakan, diamankan, dan berfungsi.

```
======================== test session starts ========================
      platform darwin -- Python 3.10.10, pytest-7.3.1, pluggy-1.0.0
      -- /Users/t/code/aws/serverless-test-samples/python-test-samples/apigw-lambda/
venv/bin/python
      cachedir: .pytest_cache
      rootdir: /Users/t/code/aws/serverless-test-samples/python-test-samples/apigw-
lambda
      plugins: mock-3.10.0
      collected 1 item


      tests/integration/test_api_gateway.py::TestApiGateway::test_api_gateway

      --> Stack outputs:

        HelloWorldApi
        = https://p7teqs3162.execute-api.us-east-2.amazonaws.com/Prod/hello/
        > API Gateway endpoint URL for Prod stage for Hello World function

        PythonTestDemo
        = arn:aws:lambda:us-east-2:123456789012:function:testing-apigw-lambda-
PythonTestDemo-iSij8evaTdxl
        > Hello World Lambda Function ARN

        PythonTestDemoIamRole
        = arn:aws:iam::123456789012:role/testing-apigw-lambda-PythonTestDemoRole-
IZELQQ9MG4HQ
        > Implicit IAM Role created for Hello World function
```

```
      --> Found API endpoint for "testing-apigw-lambda" stack...
      --> https://p7teqs3162.execute-api.us-east-2.amazonaws.com/Prod/hello/
      API Gateway response:
      amplify-dev-123456789-deployment|myapp-prod-p-loggingbucket-123456|s3-java-
bucket-123456789
      PASSED

      ========================= 1 passed in 1.53s =========================
```

Untuk pengembangan aplikasi cloud-native, pengujian di cloud memberikan manfaat berikut:

- Anda dapat menguji setiap layanan yang tersedia.
- Anda selalu menggunakan API layanan terbaru dan mengembalikan nilai.
- Lingkungan pengujian cloud sangat mirip dengan lingkungan produksi Anda.
- Pengujian dapat mencakup kebijakan keamanan, kuota layanan, konfigurasi, dan parameter spesifik infrastruktur.
- Setiap pengembang dapat dengan cepat membuat satu atau lebih lingkungan pengujian di cloud.
- Pengujian cloud meningkatkan kepercayaan kode Anda berjalan dengan benar dalam produksi.

Pengujian di cloud memang memiliki beberapa kelemahan. Penyebaran cloud biasanya memakan waktu lebih lama daripada penerapan desktop lokal.

Alat seperti AWS Serverless Application Model (AWS SAM) Accelerate, mode tontonan AWS Cloud Development Kit (AWS CDK), dan SST (pihak ke-3) mengurangi latensi ini. Alat-alat ini memantau infrastruktur dan kode Anda, lalu menyebarkan pembaruan ke lingkungan cloud Anda secara otomatis.

> **Note**
>
> Lihat cara membuat infrastruktur sebagai kode di Panduan Pengembang Tanpa Server untuk mempelajari selengkapnya AWS Serverless Application Model, CloudFormation, dan. AWS Cloud Development Kit (AWS CDK)

Tidak seperti pengujian lokal, pengujian cloud menggunakan sumber daya yang mungkin menimbulkan biaya. Lingkungan pengujian yang terisolasi dapat menambah pekerjaan untuk DevOps tim Anda, terutama di organisasi dengan kontrol akun yang ketat. Meski begitu, waktu pengembang

untuk menyiapkan lingkungan lokal yang kompleks dapat lebih mahal daripada menggunakan lingkungan cloud sekali pakai yang dibangun dengan Infrastruktur sebagai alat Kode.

Pengujian di cloud, bahkan dengan pertimbangan ini, masih merupakan cara terbaik untuk menjamin kualitas solusi tanpa server Anda.

Menguji dengan tiruan

Pengujian dengan tiruan adalah teknik di mana Anda membuat objek pengganti dalam kode Anda untuk mensimulasikan perilaku layanan cloud.

Misalnya, Anda dapat menulis tes yang menggunakan tiruan dari layanan Amazon S3. Mock mengembalikan respon set setiap kali CreateObjectmetode dipanggil. Itu tidak memanggil Amazon S3 atau titik akhir layanan lainnya.

Kerangka kerja tiruan sering menghasilkan objek tiruan untuk Anda. Beberapa kerangka kerja bersifat generik. Yang lain menargetkan AWS SDK, seperti Moto, perpustakaan Python untuk AWS layanan mengejek.

Objek tiruan berbeda dari emulator. Pengembang membuat tiruan sebagai bagian dari kode pengujian. Emulator adalah aplikasi mandiri yang mengekspos fungsionalitas yang sama dengan sistem yang mereka tiru.

Keuntungan menggunakan tiruan meliputi:

- Mocks dapat mensimulasikan layanan pihak ketiga yang berada di luar kendali aplikasi Anda, seperti penyedia API dan perangkat lunak sebagai layanan (SaaS), tanpa memerlukan akses langsung ke layanan tersebut.
- Mocks berguna untuk menguji kondisi kegagalan, terutama ketika kondisi seperti itu sulit untuk disimulasikan, seperti pemadaman layanan.
- Mock dapat memberikan pengujian lokal yang cepat setelah dikonfigurasi.
- Mocks dapat memberikan perilaku pengganti untuk hampir semua jenis objek, sehingga strategi mengejek dapat menciptakan cakupan untuk berbagai layanan yang lebih luas daripada emulator.
- Ketika fitur atau perilaku baru tersedia, pengujian tiruan dapat bereaksi lebih cepat. Dengan menggunakan kerangka kerja tiruan generik, Anda dapat mensimulasikan fitur baru segera setelah AWS SDK yang diperbarui tersedia.

Pengujian tiruan memiliki kelemahan ini:

- Mocks umumnya memerlukan upaya pengaturan dan konfigurasi yang tidak sepele, khususnya ketika mencoba menentukan nilai pengembalian dari layanan yang berbeda untuk mengejek respons dengan benar.
- Mocks ditulis, dikonfigurasi, dan harus dikelola oleh pengembang, meningkatkan tanggung jawab mereka.
- Anda mungkin perlu memiliki akses ke cloud untuk memahami API dan mengembalikan nilai layanan.
- Mocks bisa sulit dipertahankan. Saat tanda tangan API cloud yang diejek berubah, atau skema nilai pengembalian berkembang, Anda perlu memperbarui tiruan Anda. Mocks juga memerlukan pembaruan jika Anda memperluas logika aplikasi Anda untuk melakukan panggilan ke API baru.
- Pengujian yang menggunakan tiruan mungkin lolos di lingkungan desktop tetapi gagal di cloud. Hasil mungkin tidak cocok dengan API saat ini. Konfigurasi layanan dan kuota tidak dapat diuji.
- Kerangka kerja tiruan terbatas dalam menguji atau mendeteksi kebijakan atau batasan kuota AWS Identity and Access Management (IAM) and Access Management (IAM). Meskipun tiruan lebih baik dalam mensimulasikan ketika otorisasi gagal atau kuota terlampaui, pengujian tidak dapat menentukan hasil mana yang sebenarnya terjadi di lingkungan produksi.

Menguji secara lokal menggunakan AWS SAM CLI

Gunakan AWS SAMCLI untuk menguji fungsi Anda di kontainer Docker menggunakan lingkungan runtime yang sama seperti. AWS Lambda Anda dapat menguji logika fungsi dan pemrosesan peristiwa secara lokal tanpa menerapkan ke cloud. Jika fungsi Anda membuat panggilan API ke yang lain Layanan AWS, panggilan tersebut mencapai AWS sumber daya nyata.

Keuntungan pengujian dengan wadah lokal meliputi:

- Menggunakan lingkungan AWS Lambda runtime untuk pengujian yang akurat.
- Mengaktifkan iterasi pengembangan lokal yang cepat tanpa penerapan cloud.
- Mendukung debugging dengan alat pengembangan lokal yang sudah dikenal.

Pengujian dengan kontainer lokal memiliki keterbatasan ini:

- Layanan AWS panggilan dari fungsi Anda berinteraksi dengan AWS sumber daya nyata, yang mungkin menimbulkan biaya dan memengaruhi data produksi.
- Membutuhkan Docker untuk diinstal dan dijalankan secara lokal.

Pengujian dengan emulasi

Emulator adalah aplikasi yang berjalan secara lokal yang meniru AWS layanan dengan menyediakan API serupa dan mengembalikan nilai. LocalStack adalah alat emulasi populer yang menyediakan lingkungan pengembangan lokal lengkap untuk menguji integrasi layanan.

LocalStack adalah emulator AWS Cloud yang dapat Anda gunakan untuk menguji aplikasi tanpa server secara lokal. Anda dapat menguji fungsi Lambda yang terintegrasi dengan layanan seperti DynamoDB, Amazon S3, dan Amazon SQS tanpa terhubung ke layanan yang sebenarnya. AWS Anda dapat menggunakan LocalStack di AWS Toolkit for VS Code.

Keuntungan dari tes dengan emulator meliputi:

- Emulator dapat membantu iterasi dan pengujian pengembangan lokal yang cepat.
- Emulator menyediakan lingkungan yang akrab bagi pengembang yang terbiasa mengembangkan kode di lingkungan lokal. Misalnya, jika Anda terbiasa dengan pengembangan aplikasi n-tier, Anda mungkin memiliki mesin database dan server web, mirip dengan yang berjalan dalam produksi, berjalan di mesin lokal Anda untuk menyediakan kemampuan pengujian cepat, lokal, dan terisolasi.
- Emulator tidak memerlukan perubahan apa pun pada infrastruktur cloud (seperti akun cloud pengembang), sehingga mudah diterapkan dengan pola pengujian yang ada.
- Karena emulator tidak menggunakan AWS sumber daya yang sebenarnya, Anda tidak akan mendapatkan biaya tak terduga saat memulai beberapa layanan atau membiarkan beberapa sumber daya berjalan untuk waktu yang lama.

Pengujian dengan emulator memiliki kelemahan ini:

- Emulator bisa sulit diatur dan direplikasi, terutama bila digunakan dalam CI/CD jaringan pipa. Hal ini dapat meningkatkan beban kerja staf TI atau pengembang yang mengelola perangkat lunak mereka sendiri.
- Fitur dan API yang ditiru biasanya tertinggal dari pembaruan layanan. Hal ini dapat menyebabkan kesalahan karena kode yang diuji tidak cocok dengan API yang sebenarnya, dan menghambat adopsi fitur baru.
- Emulator memerlukan dukungan, pembaruan, perbaikan bug, dan peningkatan paritas fitur. Ini adalah tanggung jawab penulis emulator, yang bisa menjadi perusahaan pihak ketiga.
- Pengujian yang mengandalkan emulator mungkin memberikan hasil yang sukses secara lokal, tetapi gagal di cloud karena kebijakan keamanan produksi, konfigurasi antar-layanan, atau melebihi kuota Lambda.

Praktik terbaik

Bagian berikut memberikan rekomendasi untuk pengujian aplikasi tanpa server yang sukses.

Anda dapat menemukan contoh praktis pengujian dan otomatisasi pengujian di repositori Sampel Uji Tanpa Server.

Prioritaskan pengujian di cloud

Pengujian di cloud memberikan cakupan pengujian yang paling andal, akurat, dan lengkap. Melakukan pengujian dalam konteks cloud secara komprehensif menguji tidak hanya logika bisnis tetapi juga kebijakan keamanan, konfigurasi layanan, kuota, dan tanda tangan API terbaru dan nilai pengembalian.

Struktur kode Anda untuk pengujian

Sederhanakan pengujian dan fungsi Lambda Anda dengan Lambda-specific memisahkan kode dari logika bisnis inti Anda.

Penangan fungsi Lambda Anda harus berupa adaptor ramping yang mengambil data peristiwa dan hanya meneruskan detail yang penting bagi metode logika bisnis Anda. Dengan strategi ini, Anda dapat membungkus tes komprehensif seputar logika bisnis Anda tanpa mengkhawatirkan Lambda-specific detail. Fungsi AWS Lambda Anda seharusnya tidak memerlukan pengaturan lingkungan yang kompleks atau sejumlah besar dependensi untuk membuat dan menginisialisasi komponen yang sedang diuji.

Secara umum, Anda harus menulis handler yang mengekstrak dan memvalidasi data dari objek peristiwa dan konteks yang masuk, kemudian mengirimkan masukan itu ke metode yang melakukan logika bisnis Anda.

Mempercepat loop umpan balik pengembangan

Ada alat dan teknik untuk mempercepat loop umpan balik pengembangan. Misalnya, AWS SAM Accelerate dan mode tontonan AWS CDK mengurangi waktu yang diperlukan untuk memperbarui lingkungan cloud.

Sampel dalam repositori Sampel Uji GitHub Tanpa Server mengeksplorasi beberapa teknik ini.

Kami juga menyarankan Anda membuat dan menguji sumber daya cloud sedini mungkin selama pengembangan—tidak hanya setelah check-in ke kontrol sumber. Praktik ini memungkinkan

eksplorasi dan eksperimen yang lebih cepat saat mengembangkan solusi. Selain itu, mengotomatiskan penerapan dari mesin pengembangan membantu Anda menemukan masalah konfigurasi cloud lebih cepat dan mengurangi upaya yang sia-sia untuk pembaruan dan proses peninjauan kode.

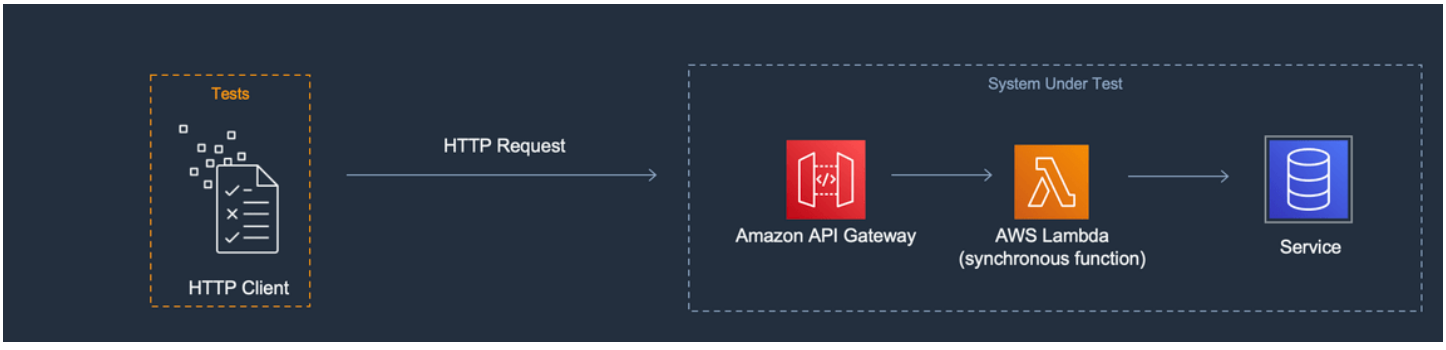
Fokus pada tes integrasi

Saat membangun aplikasi dengan Lambda, menguji komponen bersama-sama adalah praktik terbaik.

Tes yang dijalankan terhadap dua atau lebih komponen arsitektur disebut tes integrasi. Tujuan dari pengujian integrasi adalah untuk memahami tidak hanya bagaimana kode Anda mengeksekusi di seluruh komponen, tetapi bagaimana lingkungan hosting kode Anda berperilaku. End-to-end tes adalah jenis tes integrasi khusus yang memverifikasi perilaku di seluruh aplikasi.

Untuk membangun pengujian integrasi, terapkan aplikasi Anda ke lingkungan cloud. Ini dapat dilakukan dari lingkungan lokal atau melalui CI/CD pipa. Kemudian, tulis tes untuk menjalankan sistem yang sedang diuji (SUT) dan validasi perilaku yang diharapkan.

Misalnya, sistem yang diuji bisa berupa aplikasi yang menggunakan API Gateway, Lambda, dan DynamoDB. Pengujian dapat membuat panggilan HTTP sintetis ke titik akhir API Gateway dan memvalidasi bahwa respons tersebut menyertakan muatan yang diharapkan. Pengujian ini memvalidasi bahwa kode AWS Lambda sudah benar, dan setiap layanan dikonfigurasi dengan benar untuk menangani permintaan, termasuk izin IAM di antara mereka. Selanjutnya, Anda dapat merancang tes untuk menulis catatan dari berbagai ukuran untuk memverifikasi kuota layanan Anda, seperti ukuran catatan maksimal di DynamoDB, diatur dengan benar.



Buat lingkungan pengujian yang terisolasi

Pengujian di cloud biasanya membutuhkan lingkungan pengembang yang terisolasi, sehingga pengujian, data, dan peristiwa tidak tumpang tindih.

Salah satu pendekatannya adalah menyediakan setiap pengembang AWS akun khusus. Ini menghindari konflik dengan penamaan sumber daya yang dapat terjadi ketika beberapa pengembang bekerja dalam basis kode bersama, mencoba menerapkan sumber daya, atau menjalankan API.

Proses pengujian otomatis harus membuat sumber daya bernama unik untuk setiap tumpukan. Misalnya, Anda dapat mengatur skrip atau file konfigurasi TOMB sehingga perintah AWS SAM CLI sam deploy atau sam sync secara otomatis menentukan tumpukan dengan awalan unik.

Dalam beberapa kasus, pengembang berbagi AWS akun. Ini mungkin karena memiliki sumber daya di tumpukan Anda yang mahal untuk dioperasikan, atau untuk menyediakan dan mengonfigurasi. Misalnya, database mungkin dibagikan untuk mempermudah penyiapan dan menyemai data dengan benar

Jika pengembang berbagi akun, Anda harus menetapkan batasan untuk mengidentifikasi kepemilikan dan menghilangkan tumpang tindih. Salah satu cara untuk melakukannya adalah dengan mengawali nama tumpukan dengan ID pengguna pengembang. Pendekatan populer lainnya adalah mengatur tumpukan berdasarkan cabang kode. Dengan batas cabang, lingkungan terisolasi, tetapi pengembang masih dapat berbagi sumber daya, seperti database relasional. Pendekatan ini adalah praktik terbaik ketika pengembang bekerja di lebih dari satu cabang sekaligus.

Pengujian di cloud sangat berharga untuk semua fase pengujian, termasuk pengujian unit, pengujian integrasi, dan pengujian ujung ke ujung. Mempertahankan isolasi yang tepat sangat penting; tetapi Anda masih ingin lingkungan QA Anda menyerupai lingkungan produksi Anda sedekat mungkin. Untuk alasan ini, tim menambahkan proses kontrol perubahan untuk lingkungan QA.

Untuk lingkungan pra-produksi dan produksi, batasan biasanya ditarik pada tingkat akun untuk melindungi beban kerja dari masalah tetangga yang bising dan menerapkan kontrol keamanan hak istimewa paling sedikit untuk melindungi data sensitif. Beban kerja memiliki kuota. Anda tidak ingin pengujian Anda mengkonsumsi kuota yang dialokasikan untuk produksi (tetangga yang bising) atau memiliki akses ke data pelanggan. Pengujian beban adalah aktivitas lain yang harus Anda isolasi dari tumpukan produksi Anda.

Dalam semua kasus, lingkungan harus dikonfigurasi dengan peringatan dan kontrol untuk menghindari pengeluaran yang tidak perlu. Misalnya, Anda dapat membatasi jenis, tingkat, atau ukuran sumber daya yang dapat dibuat, dan mengatur peringatan email ketika perkiraan biaya melebihi ambang batas yang diberikan.

Gunakan tiruan untuk logika bisnis yang terisolasi

Kerangka kerja tiruan adalah alat yang berharga untuk menulis tes unit cepat. Mereka sangat bermanfaat ketika tes mencakup logika bisnis internal yang kompleks, seperti perhitungan atau simulasi matematika atau keuangan. Cari pengujian unit yang memiliki sejumlah besar kasus uji atau variasi input, di mana input tersebut tidak mengubah pola atau konten panggilan ke layanan cloud lainnya.

Kode yang dicakup oleh pengujian unit dengan tiruan juga harus dicakup oleh pengujian di cloud. Ini direkomendasikan karena laptop pengembang atau lingkungan mesin build dapat dikonfigurasi secara berbeda dari lingkungan produksi di cloud. Misalnya, fungsi Lambda Anda dapat menggunakan lebih banyak memori atau waktu daripada yang dialokasikan saat dijalankan dengan parameter input tertentu. Atau kode Anda mungkin menyertakan variabel lingkungan yang tidak dikonfigurasi dengan cara yang sama (atau sama sekali), dan perbedaannya dapat menyebabkan kode berperilaku berbeda atau gagal.

Manfaat tiruan kurang untuk tes integrasi, karena tingkat upaya untuk mengimplementasikan tiruan yang diperlukan meningkat dengan jumlah titik koneksi. End-to-end pengujian tidak boleh menggunakan tiruan, karena pengujian ini umumnya berurusan dengan status dan logika kompleks yang tidak dapat dengan mudah disimulasikan dengan kerangka kerja tiruan.

Terakhir, hindari menggunakan layanan cloud tiruan untuk memvalidasi implementasi panggilan layanan yang tepat. Sebagai gantinya, lakukan panggilan layanan cloud di cloud untuk memvalidasi perilaku, konfigurasi, dan implementasi fungsional.

Gunakan emulator dengan hemat

Emulator dapat nyaman untuk beberapa kasus penggunaan, misalnya, untuk tim pengembangan dengan akses internet terbatas, tidak dapat diandalkan, atau lambat. Tetapi, dalam kebanyakan keadaan, pilih untuk menggunakan emulator dengan hemat.

Dengan menghindari emulator, Anda dapat membangun dan berinovasi dengan fitur layanan terbaru dan API terbaru. Anda tidak terjebak menunggu rilis vendor untuk mencapai paritas fitur. Anda mengurangi biaya di muka dan berkelanjutan untuk pembelian dan konfigurasi pada beberapa sistem pengembangan dan membangun mesin. Selain itu, Anda menghindari masalah bahwa banyak layanan cloud tidak memiliki emulator yang tersedia. Strategi pengujian yang bergantung pada emulasi membuat tidak mungkin untuk menggunakan layanan tersebut (yang mengarah ke solusi yang berpotensi lebih mahal) atau menghasilkan kode dan konfigurasi yang tidak diuji dengan baik.

Ketika Anda menggunakan emulasi untuk pengujian, Anda masih harus menguji di cloud untuk memverifikasi konfigurasi dan untuk menguji interaksi dengan layanan cloud yang hanya dapat disimulasikan atau diejek di lingkungan yang ditiru.

Tantangan pengujian secara lokal

Saat Anda menggunakan emulator dan panggilan tiruan untuk menguji di desktop lokal Anda, Anda mungkin mengalami inkonsistensi pengujian saat kode Anda berkembang dari lingkungan ke lingkungan dalam pipeline Anda. CI/CD Pengujian unit untuk memvalidasi logika bisnis aplikasi Anda di desktop Anda mungkin tidak secara akurat menguji aspek penting dari layanan cloud.

Contoh berikut memberikan kasus yang harus diperhatikan saat menguji secara lokal dengan tiruan dan emulator:

Contoh: Event Function membuat bucket S3

Jika logika fungsi Lambda bergantung pada pembuatan bucket S3, pengujian lengkap harus mengonfirmasi bahwa Amazon S3 dipanggil dan bucket berhasil dibuat.

- Dalam pengaturan pengujian tiruan, Anda mungkin mengejek respons sukses dan berpotensi menambahkan kasus uji untuk menangani respons kegagalan.
- Dalam skenario pengujian emulasi, CreateBucketAPI mungkin dipanggil, tetapi Anda perlu menyadari bahwa identitas yang membuat panggilan lokal tidak berasal dari layanan Lambda. Identitas panggilan tidak mengambil peran keamanan seperti di cloud, jadi otentikasi placeholder digunakan sebagai gantinya, mungkin dengan peran yang lebih permisif atau identitas pengguna yang berbeda saat dijalankan di cloud.

Pengaturan tiruan dan emulasi menguji apa yang dilakukan fungsi Lambda jika memanggil Amazon S3; namun, pengujian tersebut tidak memverifikasi bahwa fungsi Lambda, seperti yang dikonfigurasi, mampu berhasil membuat bucket Amazon S3. Anda harus memastikan peran yang ditetapkan ke fungsi memiliki kebijakan keamanan terlampir yang memungkinkan fungsi untuk melakukan `s3:CreateBucket` tindakan. Jika tidak, fungsi tersebut kemungkinan gagal saat diterapkan ke lingkungan cloud.

Contoh: Fungsi Acara memproses pesan dari antrian Amazon SQS

Jika antrian Amazon SQS adalah sumber fungsi Lambda, pengujian lengkap harus memverifikasi bahwa fungsi Lambda berhasil dipanggil saat pesan dimasukkan ke dalam antrean.

Pengujian emulasi dan pengujian tiruan umumnya diatur untuk menjalankan kode fungsi Lambda secara langsung, dan untuk mensimulasikan integrasi Amazon SQS dengan meneruskan muatan peristiwa JSON (atau objek deserialisasi) sebagai input penangan fungsi.

Pengujian lokal yang mensimulasikan integrasi Amazon SQS menguji fungsi Lambda saat dipanggil oleh Amazon SQS dengan muatan tertentu, tetapi pengujian tidak memverifikasi bahwa Amazon SQS berhasil memanggil fungsi Lambda saat dikerahkan ke lingkungan cloud.

Beberapa contoh masalah konfigurasi yang mungkin Anda temui dengan Amazon SQS dan Lambda termasuk yang berikut:

- Batas waktu visibilitas Amazon SQS terlalu rendah, menghasilkan beberapa pemanggilan ketika hanya satu yang dimaksudkan.
- Peran eksekusi fungsi Lambda tidak mengizinkan membaca pesan dari antrian (melalui`sqs:ReceiveMessage`,`sqs:DeleteMessage`, atau). `sqs:GetQueueAttributes`
- Contoh peristiwa yang diteruskan ke fungsi Lambda melebihi kuota ukuran pesan Amazon SQS. Oleh karena itu, pengujian tidak valid karena Amazon SQS tidak akan pernah dapat mengirim pesan sebesar itu.

Seperti yang ditunjukkan oleh contoh-contoh ini, tes yang mencakup logika bisnis tetapi bukan konfigurasi antara layanan cloud cenderung memberikan hasil yang tidak dapat diandalkan.

Pertanyaan yang Sering Diajukan

Saya memiliki fungsi Lambda yang melakukan perhitungan dan mengembalikan hasil tanpa memanggil layanan lain. Apakah saya benar-benar perlu mengujinya di cloud?

Ya. Fungsi Lambda memiliki parameter konfigurasi yang dapat mengubah hasil pengujian. Semua kode fungsi Lambda memiliki ketergantungan pada pengaturan batas waktu dan memori, yang dapat menyebabkan fungsi gagal jika pengaturan tersebut tidak diatur dengan benar.

Kebijakan Lambda juga memungkinkan pencatatan keluaran standar ke Amazon. CloudWatch
Bahkan jika kode Anda tidak menelepon CloudWatch secara langsung, izin diperlukan untuk mengaktifkan logging. Izin yang diperlukan ini tidak dapat diejek atau ditiru secara akurat.

Bagaimana pengujian di cloud dapat membantu pengujian unit? Jika ada di cloud dan terhubung ke sumber daya lain, bukankah itu tes integrasi?

Kami mendefinisikan pengujian unit sebagai pengujian yang beroperasi pada komponen arsitektur secara terpisah, tetapi ini tidak mencegah pengujian memasukkan komponen yang mungkin memanggil layanan lain atau menggunakan beberapa komunikasi jaringan.

Banyak aplikasi tanpa server memiliki komponen arsitektur yang dapat diuji secara terpisah, bahkan di cloud. Salah satu contohnya adalah fungsi Lambda yang mengambil input, memproses data, dan mengirim pesan ke antrian Amazon SQS. Tes unit dari fungsi ini kemungkinan akan menguji apakah nilai input menghasilkan nilai tertentu yang ada dalam pesan antrian.

Pertimbangkan tes yang ditulis dengan menggunakan pola Arrange, Act, Assert:

- Atur: Alokasikan sumber daya (antrian untuk menerima pesan, dan fungsi yang sedang diuji).
- Tindakan: Panggil fungsi yang sedang diuji.
- Tegaskan: Ambil pesan yang dikirim oleh fungsi, dan validasi output.

Pendekatan pengujian tiruan akan melibatkan mengejek antrian dengan objek tiruan dalam proses, dan membuat instance dalam proses dari kelas atau modul yang berisi kode fungsi Lambda. Selama fase Assert, pesan antrian akan diambil dari objek yang diejek.

Dalam pendekatan berbasis cloud, pengujian akan membuat antrean Amazon SQS untuk keperluan pengujian, dan akan menerapkan fungsi Lambda dengan variabel lingkungan yang dikonfigurasi untuk menggunakan antrian Amazon SQS yang terisolasi sebagai tujuan keluaran. Setelah menjalankan fungsi Lambda, pengujian akan mengambil pesan dari antrian Amazon SQS.

Pengujian berbasis cloud akan menjalankan kode yang sama, menegaskan perilaku yang sama, dan memvalidasi kebenaran fungsional aplikasi. Namun, itu akan memiliki keuntungan tambahan karena dapat memvalidasi pengaturan fungsi Lambda: peran IAM, kebijakan IAM, dan batas waktu dan pengaturan memori fungsi.

Langkah dan sumber daya selanjutnya

Gunakan sumber daya berikut untuk mempelajari lebih lanjut dan mengeksplorasi contoh-contoh praktis pengujian.

Contoh implementasi

Repositori Sampel Uji Tanpa Server pada GitHub berisi contoh nyata pengujian yang mengikuti pola dan praktik terbaik yang dijelaskan dalam panduan ini. Repositori berisi kode sampel dan panduan

panduan dari proses pengujian tiruan, emulasi, dan cloud yang dijelaskan di bagian sebelumnya. Gunakan repositori ini untuk mempercepat panduan pengujian tanpa server terbaru dari. AWS

Bacaan lebih lanjut

Kunjungi Serverless Land untuk mengakses blog, video, dan pelatihan terbaru untuk teknologi tanpa AWS server.

Posting AWS blog berikut juga disarankan untuk dibaca:

- Mempercepat pengembangan tanpa server dengan AWS SAM Accelerate (posting blog)AWS
- Meningkatkan kecepatan pengembangan dengan CDK Watch (posting AWS blog)
- Integrasi layanan mengejek dengan AWS Step Functions Lokal (AWS posting blog)
- Memulai dengan menguji aplikasi tanpa server (AWS posting blog)

Alat

- AWS SAM — Menguji dan men-debug aplikasi tanpa server
- AWS SAM — Mengintegrasikan dengan tes otomatis
- Lambda - Menguji fungsi Lambda di konsol Lambda

Mengelola izin di AWS Lambda

Anda dapat menggunakan AWS Identity and Access Management (IAM) untuk mengelola izin di. AWS Lambda Ada dua kategori utama izin yang perlu Anda pertimbangkan saat bekerja dengan Fungsi Acara Lambda:

- Izin yang diperlukan fungsi Lambda Anda untuk melakukan tindakan API dan mengakses sumber daya lainnya AWS
- Izin yang diperlukan AWS pengguna dan entitas lain untuk mengakses fungsi Lambda Anda. Untuk informasi selengkapnya, lihat the section called “Izin akses (izin untuk entitas lain untuk mengakses fungsi Anda)”.

Fungsi Lambda seringkali perlu mengakses AWS sumber daya lain, dan melakukan berbagai operasi API pada sumber daya tersebut. Misalnya, Anda mungkin memiliki fungsi Lambda yang merespons peristiwa dengan memperbarui entri dalam database Amazon DynamoDB. Dalam hal ini, fungsi Anda

memerlukan izin untuk mengakses database, serta izin untuk menempatkan atau memperbarui item dalam database tersebut.

Anda menentukan izin yang dibutuhkan fungsi Lambda Anda dalam peran IAM khusus yang disebut peran eksekusi. Dalam peran ini, Anda dapat melampirkan kebijakan yang menentukan setiap izin yang diperlukan fungsi untuk mengakses AWS sumber daya lain, dan membaca dari sumber peristiwa. Setiap fungsi Lambda harus memiliki peran eksekusi. Minimal, peran eksekusi Anda harus memiliki akses ke Amazon CloudWatch karena fungsi Lambda masuk ke Log secara CloudWatch default. Anda dapat melampirkan kebijakan `AWSLambdaBasicExecutionRole` terkelola ke peran eksekusi Anda untuk memenuhi persyaratan ini.

Untuk memberikan izin lain Akun AWS, organisasi, dan layanan untuk mengakses sumber daya Lambda Anda, Anda memiliki beberapa opsi:

- Anda dapat menggunakan kebijakan berbasis identitas untuk memberi pengguna lain akses ke sumber daya Lambda Anda. Identity-based Kebijakan dapat berlaku untuk pengguna secara langsung, atau untuk grup dan peran yang terkait dengan pengguna.
- Anda dapat menggunakan kebijakan berbasis sumber daya untuk memberikan akun dan Layanan AWS izin lain untuk mengakses sumber daya Lambda Anda. Saat pengguna mencoba mengakses sumber daya Lambda, Lambda mempertimbangkan kebijakan berbasis identitas pengguna dan kebijakan berbasis sumber daya. Saat AWS layanan seperti Amazon Simple Storage Service (Amazon S3) memanggil fungsi Lambda Anda, Lambda hanya mempertimbangkan kebijakan berbasis sumber daya.
- Anda dapat menggunakan model kontrol akses berbasis atribut (ABAC) untuk mengontrol akses ke fungsi Lambda Anda. Dengan ABAC, Anda dapat melampirkan tag ke fungsi Lambda, meneruskannya dalam permintaan API tertentu, atau melampirkannya ke prinsipal IAM yang membuat permintaan. Tentukan tag yang sama dalam elemen kondisi kebijakan IAM untuk mengontrol akses fungsi.

Di AWS, ini adalah praktik terbaik untuk hanya memberikan izin yang diperlukan untuk melakukan tugas (izin hak istimewa paling sedikit). Untuk menerapkan ini di Lambda, kami sarankan memulai dengan kebijakan AWS terkelola. Anda dapat menggunakan kebijakan terkelola ini apa adanya, atau sebagai titik awal untuk menulis kebijakan Anda sendiri yang lebih ketat.

Untuk membantu Anda menyempurnakan izin untuk akses hak istimewa paling sedikit, Lambda menyediakan beberapa ketentuan tambahan yang dapat Anda sertakan dalam kebijakan Anda. Untuk informasi selengkapnya, lihat the section called “Sumber Daya dan Ketentuan”.

Untuk informasi lebih lanjut tentang IAM, lihat Panduan Pengguna IAM.

Mendefinisikan izin Lambda Event Function dengan peran eksekusi

Peran eksekusi Lambda Event Function adalah peran AWS Identity and Access Management (IAM) yang memberikan izin fungsi untuk mengakses dan sumber daya. Layanan AWS Misalnya, Anda dapat membuat peran eksekusi yang memiliki izin untuk mengirim log ke Amazon CloudWatch dan mengunggah data pelacakan AWS X-Ray. Halaman ini memberikan informasi tentang cara membuat, melihat, dan mengelola peran eksekusi fungsi Lambda.

Lambda secara otomatis mengasumsikan peran eksekusi Anda ketika Anda memanggil fungsi Anda. Anda harus menghindari pemanggilan secara manual `sts:AssumeRole` untuk mengambil peran eksekusi dalam kode fungsi Anda. Jika kasus penggunaan Anda mengharuskan peran tersebut mengasumsikan dirinya sendiri, Anda harus memasukkan peran itu sendiri sebagai prinsipal tepercaya dalam kebijakan kepercayaan peran Anda. Untuk informasi selengkapnya tentang cara mengubah kebijakan kepercayaan peran, lihat Memodifikasi kebijakan kepercayaan peran (konsol) di Panduan Pengguna IAM.

Agar Lambda dapat mengambil peran eksekusi Anda dengan benar, kebijakan kepercayaan peran harus menentukan prinsip layanan Lambda (`lambda.amazonaws.com`) sebagai layanan tepercaya.

Topik

- Membuat peran eksekusi di konsol IAM
- Menciptakan dan mengelola peran dengan AWS CLI
- Berikan akses hak istimewa paling rendah ke peran eksekusi Lambda Anda
- Melihat dan memperbarui izin dalam peran eksekusi
- Bekerja dengan kebijakan AWS terkelola dalam peran eksekusi
- Menggunakan fungsi sumber ARN untuk mengontrol perilaku akses fungsi

Membuat peran eksekusi di konsol IAM

Secara default, Lambda membuat peran eksekusi dengan izin minimal saat Anda membuat fungsi di konsol Lambda. Secara khusus, peran eksekusi ini mencakup kebijakan `AWSLambdaBasicExecutionRole` terkelola, yang memberikan izin dasar fungsi Anda untuk mencatat peristiwa ke Amazon CloudWatch Logs. Anda dapat memilih Buat peran default di bagian Izin.

Anda dapat memilih peran yang ada dengan memilih Gunakan peran lain di bagian Izin. Jika fungsi Lambda Anda memerlukan izin tambahan untuk melakukan tugas seperti memperbarui entri dalam

database Amazon DynamoDB sebagai respons terhadap peristiwa, Anda dapat membuat peran eksekusi kustom dengan izin yang diperlukan. Untuk melakukan ini, pilih Gunakan peran lain di bagian Izin, yang membuka laci tempat Anda dapat menyesuaikan izin Anda.

> **Note**
>
> Jika manajer peran diaktifkan di akun Anda, konsol Lambda akan melampirkan peran eksekusi untuk Anda. Untuk menyesuaikan izin, buat fungsi Anda, lalu edit izin fungsi untuk memilih peran lain atau menyesuaikan peran yang dilampirkan. Untuk informasi selengkapnya, lihat Membuat peran secara otomatis dengan manajer peran di Panduan Pengguna IAM.

Untuk mengonfigurasi peran eksekusi dari Konsol

1. Masukkan nama peran di bagian Rincian peran.
2. Di bagian Kebijakan, pilih Gunakan kebijakan yang ada.
3. Pilih kebijakan AWS terkelola yang ingin dilampirkan ke peran Anda. Misalnya, jika fungsi Anda perlu mengakses DynamoDB, pilih AWSLambdaDynamoDBExecutionRolekebijakan terkelola.
4. Pilih Buat peran.

Atau, saat Anda membuat fungsi di konsol Lambda, Anda dapat melampirkan peran eksekusi apa pun yang sebelumnya Anda buat ke fungsi tersebut. Jika Anda ingin melampirkan peran eksekusi baru ke fungsi yang ada, ikuti langkah-langkah dalam Memperbarui peran eksekusi fungsi.

Menciptakan dan mengelola peran dengan AWS CLI

Untuk membuat peran eksekusi dengan AWS Command Line Interface (AWS CLI), gunakan create-role perintah. Saat menggunakan perintah ini, Anda dapat menentukan kebijakan kepercayaan sebaris. Kebijakan kepercayaan peran memberikan izin kepada prinsipal yang ditentukan untuk mengambil peran tersebut. Dalam contoh berikut, Anda memberikan izin kepada kepala layanan Lambda untuk mengambil peran Anda. Perhatikan bahwa persyaratan untuk menghindari tanda kutip dalam string JSON mungkin bervariasi tergantung pada shell Anda.

```
aws iam create-role \
  --role-name lambda-ex \
```

```
  --assume-role-policy-document '{"Version": "2012-10-17",       "Statement":
[{ "Effect": "Allow", "Principal": {"Service": "lambda.amazonaws.com"}, "Action":
"sts:AssumeRole"}]}'
```

Anda juga dapat menentukan kebijakan kepercayaan untuk peran tersebut menggunakan file JSON terpisah. Dalam contoh berikut, `trust-policy.json` adalah file dalam direktori saat ini.

Example trust-policy.json

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "lambda.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

```
aws iam create-role \
  --role-name lambda-ex \
  --assume-role-policy-document file://trust-policy.json
```

Untuk menambahkan izin peran, gunakan perintah attach-policy-to-role. Perintah berikut menambahkan kebijakan `AWSLambdaBasicExecutionRole` terkelola ke peran `lambda-ex` eksekusi.

```
aws iam attach-role-policy --role-name lambda-ex --policy-arn arn:aws:iam::aws:policy/
service-role/AWSLambdaBasicExecutionRole
```

Setelah Anda membuat peran eksekusi, lampirkan ke fungsi Anda. Saat Anda membuat fungsi di konsol Lambda, Anda dapat melampirkan peran eksekusi apa pun yang sebelumnya Anda buat ke fungsi tersebut. Jika Anda ingin melampirkan peran eksekusi baru ke fungsi yang ada, ikuti langkah-langkahnyathe section called “Memperbarui peran eksekusi fungsi”.

Berikan akses hak istimewa paling rendah ke peran eksekusi Lambda Anda

Ketika Anda pertama kali membuat IAM role untuk fungsi Lambda Anda selama tahap pengembangan, Anda mungkin terkadang memberikan izin di luar apa yang diperlukan. Sebelum memublikasikan fungsi Anda di lingkungan produksi, sebagai praktik terbaik, sesuaikan kebijakan agar hanya menyertakan izin yang diperlukan. Untuk informasi selengkapnya, lihat Menerapkan izin hak istimewa terkecil di Panduan Pengguna IAM.

Menggunakan IAM Access Analyzer untuk membantu mengidentifikasi izin yang diperlukan untuk kebijakan peran eksekusi IAM. IAM Access Analyzer meninjau AWS CloudTrail log Anda selama rentang tanggal yang Anda tentukan dan menghasilkan templat kebijakan hanya dengan izin yang digunakan fungsi selama waktu tersebut. Anda dapat menggunakan templat untuk membuat kebijakan terkelola dengan izin terperinci, lalu melampirkannya ke IAM role. Dengan begitu, Anda hanya memberikan izin yang diperlukan peran untuk berinteraksi dengan AWS sumber daya untuk kasus penggunaan spesifik Anda.

Untuk informasi selengkapnya, lihat Menghasilkan kebijakan berdasarkan aktivitas akses di Panduan Pengguna IAM.

Melihat dan memperbarui izin dalam peran eksekusi

Topik ini mencakup bagaimana Anda dapat melihat dan memperbarui peran eksekusi fungsi Anda.

Topik

- Melihat peran eksekusi fungsi
- Memperbarui peran eksekusi fungsi

Melihat peran eksekusi fungsi

Untuk melihat peran eksekusi fungsi, gunakan konsol Lambda.

Untuk melihat peran eksekusi fungsi (konsol)

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih nama sebuah fungsi.
3. Pilih Konfigurasi, lalu pilih Izin.
4. Di bawah Peran eksekusi, Anda dapat melihat peran yang saat ini digunakan sebagai peran eksekusi fungsi. Untuk kenyamanan, Anda dapat melihat semua sumber daya dan tindakan

yang dapat diakses fungsi di bawah bagian Ringkasan sumber daya. Anda juga dapat memilih layanan dari daftar dropdown untuk melihat semua izin yang terkait dengan layanan tersebut.

Memperbarui peran eksekusi fungsi

Anda dapat menambahkan atau menghapus izin dari peran eksekusi fungsi kapan saja, atau mengonfigurasi fungsi Anda untuk menggunakan peran yang berbeda. Jika fungsi Anda memerlukan akses ke layanan atau sumber daya lain, Anda harus menambahkan izin yang diperlukan ke peran eksekusi.

Saat Anda menambahkan izin ke fungsi Anda, lakukan pembaruan sepele ke kode atau konfigurasinya juga. Ini memaksa instance menjalankan fungsi Anda, yang memiliki kredensialnya yang sudah ketinggalan zaman, untuk berhenti dan diganti.

Untuk memperbarui peran eksekusi fungsi, Anda dapat menggunakan konsol Lambda.

Untuk memperbarui peran eksekusi fungsi (konsol)

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih nama sebuah fungsi.
3. Pilih Konfigurasi, lalu pilih Izin.
4. Di bawah Peran eksekusi, pilih Edit.
5. Jika Anda ingin memperbarui fungsi Anda untuk menggunakan peran yang berbeda sebagai peran eksekusi, pilih peran baru di menu tarik-turun di bawah Peran yang ada.

   > **Note**
   >
   > Jika Anda ingin memperbarui izin dalam peran eksekusi yang ada, Anda hanya dapat melakukannya di konsol AWS Identity and Access Management (IAM).

   Jika Anda ingin membuat peran baru untuk digunakan sebagai peran eksekusi, pilih Buat peran baru dari templat AWS kebijakan di bawah Peran eksekusi. Kemudian, masukkan nama untuk peran baru Anda di bawah Nama peran, dan tentukan kebijakan apa pun yang ingin Anda lampirkan ke peran baru di bawah Templat kebijakan.
6. Pilih Simpan.

Bekerja dengan kebijakan AWS terkelola dalam peran eksekusi

Kebijakan AWS terkelola berikut memberikan izin yang diperlukan untuk menggunakan fitur Lambda.

Perubahan	Deskripsi	Tanggal
AWSLambdaMSKExecutionPeran — Lambda menambahkan izin kafka:DescribeCluster V2 ke kebijakan ini.	`AWSLambdaMSKExecutionRole` memberikan izin untuk membaca dan mengakses catatan dari kluster Amazon Managed Streaming for Apache Kafka (Amazon MSK), mengelola antarmuka ENIs jaringan elastis (), dan menulis ke Log. CloudWatch	Juni 17, 2022
AWSLambdaBasicExecutionRole— Lambda mulai melacak perubahan pada kebijakan ini.	`AWSLambdaBasicExecutionRole` memberikan izin untuk mengunggah log ke. CloudWatch	14 Februari 2022
AWSLambdaDBExecutionPeran Dynamo — Lambda mulai melacak perubahan pada kebijakan ini.	`AWSLambdaDynamoDBExecutionRole` memberikan izin untuk membaca catatan dari aliran Amazon DynamoDB dan menulis ke Log. CloudWatch	14 Februari 2022
AWSLambdaKinesisExecutionRole— Lambda mulai melacak perubahan pada kebijakan ini.	`AWSLambdaKinesisExecutionRole` memberikan izin untuk membaca peristiwa dari aliran data Amazon Kinesis dan menulis ke Log. CloudWatch	14 Februari 2022
AWSLambdaMSKExecutionPeran — Lambda mulai	`AWSLambdaMSKExecutionRole` memberikan	14 Februari 2022

Perubahan	Deskripsi	Tanggal
melacak perubahan pada kebijakan ini.	izin untuk membaca dan mengakses catatan dari kluster Amazon Managed Streaming for Apache Kafka (Amazon MSK), mengelola antarmuka ENIs jaringan elastis (), dan menulis ke Log. CloudWatch	
AWSLambdaSQSQueueExecutionRole— Lambda mulai melacak perubahan pada kebijakan ini.	`AWSLambdaSQSQueueExecutionRole` memberikan izin untuk membaca pesan dari antrian Amazon Simple Queue Service (Amazon SQS) dan menulis ke Log. CloudWatch	14 Februari 2022
AWSLambdaVPCAccessExecutionRole— Lambda mulai melacak perubahan pada kebijakan ini.	`AWSLambdaVPCAccessExecutionRole` memberikan izin untuk mengelola ENIs dalam VPC Amazon dan menulis ke Log. CloudWatch	14 Februari 2022
AWSXRayDaemonWriteAccess— Lambda mulai melacak perubahan pada kebijakan ini.	`AWSXRayDaemonWriteAccess` memberikan izin untuk mengunggah data jejak ke X-Ray.	14 Februari 2022
CloudWatchLambdaInsightsExecutionRolePolicy— Lambda mulai melacak perubahan pada kebijakan ini.	`CloudWatchLambdaInsightsExecutionRolePolicy` memberikan izin untuk menulis metrik runtime ke Lambda Insights. CloudWatch	14 Februari 2022

Perubahan	Deskripsi	Tanggal
AmazonS3 - ObjectLambdaExecutionRolePolicy Lambda mulai melacak perubahan pada kebijakan ini.	`AmazonS3ObjectLambdaExecutionRolePolicy` memberikan izin untuk berinteraksi dengan objek Amazon Simple Storage Service (Amazon S3) Lambda dan menulis ke Log. CloudWatch	14 Februari 2022

Untuk beberapa fitur, konsol Lambda mencoba menambahkan izin yang hilang ke peran eksekusi Anda dalam kebijakan yang dikelola pelanggan. Kebijakan ini dapat menjadi beberapa. Untuk menghindari pembuatan kebijakan tambahan, tambahkan kebijakan yang dikelola AWS yang relevan ke peran eksekusi Anda sebelum mengaktifkan fitur.

Saat Anda menggunakan pemetaan sumber peristiwa untuk mengaktifkan fungsi Anda, Lambda menggunakan peran eksekusi untuk membaca data peristiwa. Misalnya, pemetaan sumber peristiwa untuk Kinesis membaca peristiwa dari aliran data dan mengirimnya ke fungsi Anda dalam beberapa batch.

Saat layanan mengambil peran di akun Anda, Anda dapat menyertakan kunci konteks kondisi `aws:SourceArn` global `aws:SourceAccount` dan global dalam kebijakan kepercayaan peran untuk membatasi akses ke peran hanya pada permintaan yang dihasilkan oleh sumber daya yang diharapkan. Untuk informasi lebih lanjut, lihat Pencegahan wakil kebingungan lintas layanan untuk AWS Security Token Service.

Selain kebijakan AWS terkelola, konsol Lambda menyediakan templat untuk membuat kebijakan khusus dengan izin untuk kasus penggunaan tambahan. Saat Anda membuat fungsi di konsol Lambda, Anda dapat memilih untuk membuat peran eksekusi baru dengan izin dari satu templat atau lebih. Templat ini juga diterapkan secara otomatis ketika Anda membuat fungsi dari cetak biru, atau ketika Anda mengonfigurasi opsi yang memerlukan akses ke layanan lain. Contoh templat tersedia dalam repositori GitHub panduan ini.

Menggunakan fungsi sumber ARN untuk mengontrol perilaku akses fungsi

Biasanya kode Fungsi Peristiwa Lambda Anda membuat permintaan API ke yang lain. Layanan AWS Untuk membuat permintaan ini, Lambda menghasilkan kumpulan kredensyal sementara

dengan mengasumsikan peran eksekusi fungsi Anda. Kredensyal ini tersedia sebagai variabel lingkungan selama pemanggilan fungsi Anda. Saat bekerja dengan AWS SDK, Anda tidak perlu memberikan kredensyal untuk SDK secara langsung dalam kode. Secara default, rantai penyedia kredensyal secara berurutan memeriksa setiap tempat di mana Anda dapat menyetel kredensyal dan memilih yang pertama tersedia—biasanya variabel lingkungan (,, dan). `AWS_ACCESS_KEY_ID` `AWS_SECRET_ACCESS_KEY` `AWS_SESSION_TOKEN`

Lambda menyuntikkan ARN fungsi sumber ke dalam konteks kredensyal jika permintaan tersebut adalah permintaan AWS API yang berasal dari dalam lingkungan eksekusi Anda. Lambda juga menyuntikkan ARN fungsi sumber untuk AWS permintaan API berikut yang dibuat Lambda atas nama Anda di luar lingkungan eksekusi Anda:

Layanan	Tindakan	Alasan
CloudWatch Log	`CreateLogGroup` , `CreateLogStream` , `PutLogEvents`	Untuk menyimpan log ke grup CloudWatch log Log
X-Ray	`PutTraceSegments`	Untuk mengirim data jejak ke X-Ray
Amazon EFS	`ClientMount`	Untuk menghubungkan fungsi Anda ke sistem file Amazon Elastic File System (Amazon EFS)

Panggilan AWS API lain yang dilakukan Lambda di luar lingkungan eksekusi Anda atas nama Anda menggunakan peran eksekusi yang sama tidak mengandung ARN fungsi sumber. Contoh panggilan API tersebut di luar lingkungan eksekusi meliputi:

- Panggilan ke AWS Key Management Service (AWS KMS) untuk mengenkripsi dan mendekripsi variabel lingkungan Anda secara otomatis.
- Panggilan ke Amazon Elastic Compute Cloud (Amazon EC2) untuk membuat antarmuka jaringan elastis (ENI) untuk suatu fungsi. VPC-enabled
- Panggilan ke Layanan AWS, seperti Amazon Simple Queue Service (Amazon SQS), untuk membaca dari sumber peristiwa yang disiapkan sebagai pemetaan sumber peristiwa.

Dengan fungsi sumber ARN dalam konteks kredensyal, Anda dapat memverifikasi apakah panggilan ke sumber daya Anda berasal dari kode fungsi Lambda tertentu. Untuk memverifikasi ini, gunakan kunci `lambda:SourceFunctionArn` kondisi dalam kebijakan berbasis identitas IAM atau kebijakan kontrol layanan (SCP).

> **Note**
>
> Anda tidak dapat menggunakan kunci `lambda:SourceFunctionArn` kondisi dalam kebijakan berbasis sumber daya.

Dengan kunci kondisi ini dalam kebijakan berbasis identitas atau SCP, Anda dapat menerapkan kontrol keamanan untuk tindakan API yang dibuat kode fungsi Anda ke yang lain. Layanan AWS Ini memiliki beberapa aplikasi keamanan utama, seperti membantu Anda mengidentifikasi sumber kebocoran kredensyal.

> **Note**
>
> Kunci `lambda:SourceFunctionArn` kondisi berbeda dari tombol `lambda:FunctionArn` dan `aws:SourceArn` kondisi. Kunci `lambda:FunctionArn` kondisi hanya berlaku untuk pemetaan sumber peristiwa dan membantu menentukan fungsi mana yang dapat dipanggil sumber acara Anda. Kunci `aws:SourceArn` kondisi hanya berlaku untuk kebijakan di mana fungsi Lambda Anda adalah sumber daya target, dan membantu menentukan sumber lain Layanan AWS dan sumber daya mana yang dapat menjalankan fungsi tersebut. Kunci `lambda:SourceFunctionArn` kondisi dapat diterapkan pada kebijakan berbasis identitas atau SCP apa pun untuk menentukan fungsi Lambda tertentu yang memiliki izin untuk melakukan panggilan API tertentu ke sumber daya lain. AWS

Untuk digunakan `lambda:SourceFunctionArn` dalam polis Anda, sertakan sebagai syarat dengan salah satu operator kondisi ARN. Nilai kunci harus berupa ARN yang valid.

Misalnya, misalkan kode fungsi Lambda Anda membuat `s3:PutObject` panggilan yang menargetkan bucket Amazon S3 tertentu. Anda mungkin ingin mengizinkan hanya satu fungsi Lambda tertentu untuk `s3:PutObject` mengakses bucket itu. Dalam hal ini, peran eksekusi fungsi Anda harus memiliki kebijakan yang dilampirkan yang terlihat seperti ini:

Example kebijakan yang memberikan akses Fungsi Peristiwa tertentu ke sumber daya Amazon S3

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ExampleSourceFunctionArn",
            "Effect": "Allow",
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::lambda_bucket/*",
            "Condition": {
                "ArnEquals": {
                    "lambda:SourceFunctionArn": "arn:aws:lambda:us-
east-1:123456789012:function:source_lambda"
                }
            }
        }
    ]
}
```

Kebijakan ini hanya mengizinkan `s3:PutObject` akses jika sumbernya adalah fungsi Lambda dengan ARN. `arn:aws:lambda:us-east-1:123456789012:function:source_lambda` Kebijakan ini tidak mengizinkan `s3:PutObject` akses ke identitas panggilan lainnya. Ini benar bahkan jika fungsi atau entitas yang berbeda membuat `s3:PutObject` panggilan dengan peran eksekusi yang sama.

> **Note**
>
> Kunci `lambda:SourceFunctionARN` kondisi tidak mendukung versi fungsi Lambda atau alias fungsi. Jika Anda menggunakan ARN untuk versi atau alias fungsi tertentu, fungsi Anda tidak akan memiliki izin untuk mengambil tindakan yang Anda tentukan. Pastikan untuk menggunakan ARN yang tidak memenuhi syarat untuk fungsi Anda tanpa versi atau akhiran alias.

Anda juga dapat menggunakannya `lambda:SourceFunctionArn` di SCP. Misalnya, Anda ingin membatasi akses ke bucket ke kode fungsi Lambda tunggal atau panggilan dari Amazon Virtual Private Cloud (VPC) tertentu. SCP berikut menggambarkan hal ini.

Example kebijakan yang menolak akses ke Amazon S3 dalam kondisi tertentu

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "s3:*"
            ],
            "Resource": "arn:aws:s3:::lambda_bucket/*",
            "Effect": "Deny",
            "Condition": {
                "StringNotEqualsIfExists": {
                    "aws:SourceVpc": [
                        "vpc-12345678"
                    ]
                }
            }
        },
        {
            "Action": [
                "s3:*"
            ],
            "Resource": "arn:aws:s3:::lambda_bucket/*",
            "Effect": "Deny",
            "Condition": {
                "ArnNotEqualsIfExists": {
                    "lambda:SourceFunctionArn": "arn:aws:lambda:us-
east-1:123456789012:function:source_lambda"
                }
            }
        }
    ]
}
```

Kebijakan ini menolak semua tindakan S3 kecuali berasal dari fungsi Lambda tertentu dengan ARN`arn:aws:lambda:*:123456789012:function:source_lambda`, atau kecuali berasal dari VPC yang ditentukan. `StringNotEqualsIfExists`Operator memberitahu IAM untuk memproses kondisi ini hanya jika `aws:SourceVpc` kunci ada dalam permintaan. Demikian pula, IAM menganggap `ArnNotEqualsIfExists` operator hanya jika `lambda:SourceFunctionArn` ada.

Pemberian lainnya AWS entitas akses ke Fungsi Acara Anda

Untuk memberikan izin lain Akun AWS, organisasi, dan layanan untuk mengakses sumber daya Lambda Anda, Anda memiliki beberapa opsi:

- Anda dapat menggunakan kebijakan berbasis identitas untuk memberi pengguna lain akses ke sumber daya Lambda Anda. Identity-based Kebijakan dapat berlaku untuk pengguna secara langsung, atau untuk grup dan peran yang terkait dengan pengguna.
- Anda dapat menggunakan kebijakan berbasis sumber daya untuk memberikan akun dan Layanan AWS izin lain untuk mengakses sumber daya Lambda Anda. Saat pengguna mencoba mengakses sumber daya Lambda, Lambda mempertimbangkan kebijakan berbasis identitas pengguna dan kebijakan berbasis sumber daya. Saat AWS layanan seperti Amazon Simple Storage Service (Amazon S3) memanggil fungsi Lambda Anda, Lambda hanya mempertimbangkan kebijakan berbasis sumber daya.
- Anda dapat menggunakan model kontrol akses berbasis atribut (ABAC) untuk mengontrol akses ke fungsi Lambda Anda. Dengan ABAC, Anda dapat melampirkan tag ke fungsi Lambda, meneruskannya dalam permintaan API tertentu, atau melampirkannya ke prinsipal IAM yang membuat permintaan. Tentukan tag yang sama dalam elemen kondisi kebijakan IAM untuk mengontrol akses fungsi.

Untuk membantu Anda menyempurnakan izin untuk akses hak istimewa paling sedikit, Lambda menyediakan beberapa ketentuan tambahan yang dapat Anda sertakan dalam kebijakan Anda. Lihat informasi yang lebih lengkap di the section called “Sumber Daya dan Ketentuan”.

Identity-based Kebijakan IAM untuk Lambda

Anda dapat menggunakan kebijakan berbasis identitas di AWS Identity and Access Management (IAM) untuk memberikan pengguna di akun Anda akses ke Lambda. Identity-based kebijakan dapat berlaku untuk pengguna, grup pengguna, atau peran. Anda juga dapat memberi pengguna di akun lain izin untuk berperan di akun Anda dan mengakses sumber daya Lambda Anda.

Lambda menyediakan kebijakan AWS terkelola yang memberikan akses ke tindakan API Lambda dan, dalam beberapa kasus, akses ke AWS layanan lain yang digunakan untuk mengembangkan dan mengelola sumber daya Lambda. Lambda memperbarui kebijakan terkelola ini sesuai kebutuhan, untuk memastikan bahwa pengguna Anda memiliki akses ke fitur baru ketika dirilis.

- AWS Lambda_FullAccess — Memberikan akses penuh ke tindakan Lambda dan layanan AWS lain yang digunakan untuk mengembangkan dan memelihara sumber daya Lambda.

- AWS Lambda_ReadOnlyAccess — Memberikan akses hanya-baca ke sumber daya Lambda.
- AWSLambdaRole— Memberikan izin untuk memanggil Fungsi Acara.

AWS kebijakan terkelola memberikan izin untuk tindakan API tanpa membatasi fungsi atau lapisan Lambda yang dapat dimodifikasi pengguna. Untuk kontrol yang lebih cermat, Anda dapat membuat kebijakan Anda sendiri yang membatasi ruang lingkup izin pengguna. Untuk informasi tentang pemberian akses lapisan, lihatthe section called “Akses lapisan”.

Topik

- Memberikan pengguna akses ke Fungsi Acara
- Memberikan pengguna akses ke lapisan Lambda

Memberikan pengguna akses ke Fungsi Acara

Gunakan kebijakan berbasis identitas untuk mengizinkan pengguna, grup pengguna, atau peran menjalankan operasi di Fungsi Peristiwa Lambda.

> **Note**
>
> Untuk fungsi yang didefinisikan sebagai image kontainer, izin pengguna untuk mengakses gambar harus dikonfigurasi di Amazon Elastic Container Registry (Amazon ECR). Sebagai contoh, lihat Kebijakan repositori Amazon ECR.

Berikut ini adalah contoh kebijakan izin dengan lingkup terbatas. Ini memungkinkan pengguna membuat dan mengelola fungsi Lambda yang diberi nama dengan awalan yang ditentukan (`intern-`), dan dikonfigurasi dengan peran eksekusi yang ditunjuk.

Example Kebijakan pengembangan fungsi

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ReadOnlyPermissions",
            "Effect": "Allow",
```

```
            "Action": [
                "lambda:GetAccountSettings",
                "lambda:GetEventSourceMapping",
                "lambda:GetFunction",
                "lambda:GetFunctionConfiguration",
                "lambda:GetFunctionCodeSigningConfig",
                "lambda:GetFunctionConcurrency",
                "lambda:ListEventSourceMappings",
                "lambda:ListFunctions",
                "lambda:ListTags",
                "iam:ListRoles"
            ],
            "Resource": "*"
        },
        {
            "Sid": "DevelopFunctions",
            "Effect": "Allow",
            "NotAction": [
                "lambda:AddPermission",
                "lambda:PutFunctionConcurrency"
            ],
            "Resource": "arn:aws:lambda:*:*:function:intern-*"
        },
        {
            "Sid": "DevelopEventSourceMappings",
            "Effect": "Allow",
            "Action": [
                "lambda:DeleteEventSourceMapping",
                "lambda:UpdateEventSourceMapping",
                "lambda:CreateEventSourceMapping"
            ],
            "Resource": "*",
            "Condition": {
                "ArnLike": {
                    "lambda:FunctionArn": "arn:aws:lambda:*:*:function:intern-*"
                }
            }
        },
        {
            "Sid": "PassExecutionRole",
            "Effect": "Allow",
            "Action": [
                "iam:ListRolePolicies",
                "iam:ListAttachedRolePolicies",
```

```
                "iam:GetRole",
                "iam:GetRolePolicy",
                "iam:PassRole",
                "iam:SimulatePrincipalPolicy"
            ],
            "Resource": "arn:aws:iam::*:role/intern-lambda-execution-role"
        },
        {
            "Sid": "ViewLogs",
            "Effect": "Allow",
            "Action": [
                "logs:*"
            ],
            "Resource": "arn:aws:logs:*:*:log-group:/aws/lambda/intern-*"
        }
    ]
}
```

Izin dalam kebijakan disusun menjadi pernyataan berdasarkan sumber daya dan kondisi yang mereka dukung.

- `ReadOnlyPermissions` – Konsol Lambda menggunakan izin ini saat Anda menelusuri dan melihat fungsi. Mereka tidak mendukung pola atau kondisi sumber daya.

```
        "Action": [
            "lambda:GetAccountSettings",
            "lambda:GetEventSourceMapping",
            "lambda:GetFunction",
            "lambda:GetFunctionConfiguration",
            "lambda:GetFunctionCodeSigningConfig",
            "lambda:GetFunctionConcurrency",
            "lambda:ListEventSourceMappings",
            "lambda:ListFunctions",
            "lambda:ListTags",
            "iam:ListRoles"
        ],
        "Resource": "*"
```

- `DevelopFunctions`— Gunakan tindakan Lambda apa pun yang beroperasi pada fungsi yang diawali dengan`intern-`, kecuali dan. `AddPermission` `PutFunctionConcurrency`

`AddPermission`memodifikasi kebijakan berbasis sumber daya pada fungsi dan dapat memiliki implikasi keamanan. `PutFunctionConcurrency`cadangan kapasitas penskalaan untuk suatu fungsi dan dapat mengambil kapasitas dari fungsi lain.

```
"NotAction": [
    "lambda:AddPermission",
    "lambda:PutFunctionConcurrency"
],
"Resource": "arn:aws:lambda:*:*:function:intern-*"
```

- `DevelopEventSourceMappings` – Mengelola pemetaan sumber peristiwa pada fungsi yang diawali dengan `intern-`. Tindakan ini beroperasi pada pemetaan sumber peristiwa, tetapi Anda dapat membatasinya menurut fungsi dengan syarat.

```
"Action": [
    "lambda:DeleteEventSourceMapping",
    "lambda:UpdateEventSourceMapping",
    "lambda:CreateEventSourceMapping"
],
"Resource": "*",
"Condition": {
    "StringLike": {
        "lambda:FunctionArn": "arn:aws:lambda:*:*:function:intern-*"
    }
}
```

- `PassExecutionRole` – Lihat dan teruskan hanya peran bernama `intern-lambda-execution-role`, yang harus dibuat dan dikelola oleh pengguna dengan izin IAM. `PassRole` digunakan saat Anda menetapkan peran eksekusi ke fungsi.

```
"Action": [
    "iam:ListRolePolicies",
    "iam:ListAttachedRolePolicies",
    "iam:GetRole",
    "iam:GetRolePolicy",
    "iam:PassRole",
    "iam:SimulatePrincipalPolicy"
],
```

```
            "Resource": "arn:aws:iam::*:role/intern-lambda-execution-role"
```

- `ViewLogs`— Gunakan CloudWatch Log untuk melihat log untuk fungsi yang diawali dengan`intern-`.

```
            "Action": [
                "logs:*"
            ],
            "Resource": "arn:aws:logs:*:*:log-group:/aws/lambda/intern-*"
```

Kebijakan ini memungkinkan pengguna untuk memulai dengan Lambda, tanpa menempatkan sumber daya pengguna lain dalam risiko. Itu tidak memungkinkan pengguna untuk mengonfigurasi fungsi yang akan dipicu oleh atau memanggil AWS layanan lain, yang memerlukan izin IAM yang lebih luas. Ini juga tidak termasuk izin untuk layanan yang tidak mendukung kebijakan lingkup terbatas, seperti CloudWatch dan. X-Ray Gunakan kebijakan hanya-baca untuk layanan ini agar pengguna dapat mengakses data metrik dan jejak.

Saat Anda mengonfigurasi pemicu untuk fungsi Anda, Anda memerlukan akses untuk menggunakan AWS layanan yang memanggil fungsi Anda. Misalnya, untuk mengonfigurasi pemicu Amazon S3, Anda memerlukan izin untuk menggunakan tindakan Amazon S3 yang mengelola pemberitahuan bucket. Banyak dari izin ini disertakan dalam kebijakan Lambda_FullAccess terkelola AWS.

Memberikan pengguna akses ke lapisan Lambda

Gunakan kebijakan berbasis identitas untuk memungkinkan pengguna, grup pengguna, atau peran melakukan operasi pada lapisan Lambda. Kebijakan berikut memberikan izin pengguna untuk membuat lapisan dan menggunakan mereka dengan fungsi. Pola sumber daya memungkinkan pengguna untuk bekerja di setiap Wilayah AWS dan dengan versi lapisan apa pun, selama nama lapisan dimulai dengan`test-`.

Example kebijakan pengembangan lapisan

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
```

```
            "Sid": "PublishLayers",
            "Effect": "Allow",
            "Action": [
                "lambda:PublishLayerVersion"
            ],
            "Resource": "arn:aws:lambda:*:*:layer:test-*"
        },
        {
            "Sid": "ManageLayerVersions",
            "Effect": "Allow",
            "Action": [
                "lambda:GetLayerVersion",
                "lambda:DeleteLayerVersion"
            ],
            "Resource": "arn:aws:lambda:*:*:layer:test-*:*"
        }
    ]
}
```

Anda juga dapat menerapkan penggunaan lapisan selama pembuatan fungsi dan konfigurasi dengan kondisi `lambda:Layer`. Misalnya, Anda dapat mencegah pengguna dari menggunakan lapisan yang diterbitkan oleh akun lain. Kebijakan berikut ini menambahkan kondisi ke `CreateFunction` dan `UpdateFunctionConfiguration` tindakan untuk mewajibkan setiap lapisan yang ditentukan berasal dari akun `123456789012`.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ConfigureFunctions",
            "Effect": "Allow",
            "Action": [
                "lambda:CreateFunction",
                "lambda:UpdateFunctionConfiguration"
            ],
            "Resource": "*",
            "Condition": {
                "ForAllValues:StringLike": {
                    "lambda:Layer": [
```

```
                    "arn:aws:lambda:*:123456789012:layer:*:*"
                ]
            }
        }
    }
  ]
}
```

Untuk memastikan kondisi berlaku, verifikasikan bahwa tidak ada pernyataan lain yang memberikan izin pengguna atas tindakan ini.

Bekerja dengan kebijakan berbasis sumber daya di Lambda

Dengan kebijakan izin berbasis sumber daya, Anda dapat memberikan akses lain, pengguna, organisasi Akun AWS, dan akses Layanan AWS ke Fungsi Peristiwa Lambda Anda. Kebijakan berbasis sumber daya adalah dokumen JSON yang berisi satu atau beberapa pernyataan. Setiap pernyataan mendefinisikan yang berikut:

- `Principal`: Entitas yang ingin Anda berikan izin (yang lain Layanan AWS, peran IAM atau pengguna, atau lainnya) Akun AWS
- `Action`: Daftar tindakan API yang ingin Anda izinkan atau tolak untuk prinsipal yang ditentukan
- `Effect`: Apakah Anda ingin mengizinkan atau menolak kemampuan utama untuk menggunakan tindakan API yang dipilih
- `Resource`: Fungsi, versi, atau alias Lambda yang Anda inginkan untuk menerapkan pernyataan (Anda juga dapat menggunakan karakter wildcard untuk menentukan semua versi dan alias fungsi Anda)

Anda juga dapat menggunakan elemen opsional seperti `Sid` (pengidentifikasi pernyataan) dan `Condition` (kondisi logis untuk kontrol akses berbutir halus). Untuk daftar lengkap elemen kebijakan yang didukung, lihat referensi elemen kebijakan IAM JSON di AWS Identity and Access Management Panduan Pengguna.

Menambahkan izin berbasis sumber daya ke Fungsi Acara

Anda dapat menambahkan izin berbasis sumber daya ke fungsi Lambda Anda menggunakan dua metode:

- Kebijakan JSON lengkap — Gunakan konsol Lambda AWS CLI, atau PutResourcePolicytindakan API untuk menambahkan dokumen kebijakan JSON yang lengkap. Dengan kebijakan JSON lengkap, Anda dapat menggunakan rangkaian lengkap kunci kondisi global IAM, menambahkan beberapa pernyataan dengan beberapa prinsip, dan membuat pernyataan eksplisit. `Deny` Ukuran maksimum untuk kebijakan berbasis sumber daya JSON adalah 20 KB.
- Izin individual — Gunakan tindakan konsol atau AddPermissionAPI untuk menambahkan `Allow` pernyataan tunggal. Izin individual hanya mendukung sekumpulan kunci kondisi terbatas (`aws:SourceArn`,`aws:SourceAccount`, dan`aws:PrincipalOrgID`).

Sebaiknya Anda menentukan kebijakan JSON lengkap untuk menambahkan izin berbasis sumber daya ke fungsi Anda. Membuat kebijakan JSON yang lengkap memberi Anda lebih banyak fleksibilitas dan kontrol halus atas izin Anda.

> **Important**
>
> Menggunakan `put-resource-policy` menggantikan kebijakan berbasis sumber daya yang ada pada sumber daya. Jika sumber daya sudah memiliki izin yang ditentukan`add-permission`, `put-resource-policy` timpa. Gunakan `get-resource-policy` untuk mengambil kebijakan yang ada sebelum membuat perubahan.

Izin yang diperlukan

Untuk menggunakan tindakan`PutResourcePolicy`,`GetResourcePolicy`, dan `DeleteResourcePolicy` API, Anda memerlukan izin IAM berikut:

Tindakan API	Izin yang diperlukan
PutResourcePolicy	`lambda:PutResourcePolicy` , `lambda:AddPermission` , dan `lambda:RemovePermission`
GetResourcePolicy	`lambda:GetResourcePolicy` dan `lambda:GetPolicy`
DeleteResourcePolicy	`lambda:DeleteResourcePolicy` dan `lambda:RemovePermission`

Console

Untuk membuat kebijakan berbasis sumber daya JSON lengkap

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih fungsi yang ingin Anda berikan akses dan kemudian pilih tab Konfigurasi.
3. Pilih Izin.
4. Di panel pernyataan Resource-based kebijakan, pilih Edit. Tindakan ini membuka editor kebijakan JSON.
5. Tambahkan pernyataan ke kebijakan fungsi Anda. Anda dapat memilih tindakan API, menambahkan prinsip dari berbagai jenis (layanan, akun, peran IAM), dan menambahkan kunci kondisi untuk mengontrol akses.
6. Pilih Simpan.

Anda juga dapat mengedit kebijakan berbasis sumber daya fungsi Anda secara langsung di panel Kebijakan.

AWS CLI

Untuk menambahkan kebijakan berbasis sumber daya JSON lengkap

Untuk menambahkan kebijakan JSON lengkap ke fungsi Anda, gunakan `put-resource-policy` AWS CLI perintah.

Perintah contoh berikut menambahkan kebijakan berbasis sumber daya ke fungsi Anda menggunakan kebijakan yang ditentukan dalam file bernama `policy.json` di mesin lokal Anda. Jalankan perintah dari direktori yang berisi file. `resource-arn`Dapat menentukan versi fungsi atau alias, atau Anda dapat menggunakan fungsi ARN yang tidak memenuhi syarat untuk menerapkan kebijakan ke seluruh fungsi.

```
aws lambda put-resource-policy --resource-arn arn:aws:lambda:us-
east-2:123456789012:function:my-function \
--policy file://policy.json
```

Lambda APIs

Untuk menambahkan kebijakan izin JSON lengkap ke fungsi Anda, gunakan tindakan PutResourcePolicyAPI. Anda juga dapat menghapus kebijakan fungsi menggunakan

DeleteResourcePolicytindakan, atau mengambil kebijakan yang saat ini dilampirkan ke fungsi menggunakan GetResourcePolicytindakan.

Untuk menambahkan izin individual ke kebijakan fungsi, gunakan tindakan AddPermissionAPI.

Melihat kebijakan berbasis sumber daya fungsi

Console

Untuk menampilkan kebijakan fungsi berbasis sumber daya

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih fungsi.
3. Pilih Konfigurasi, lalu pilih Izin.
4. Gulir ke bawah ke pernyataan Resource-based kebijakan untuk melihat kebijakan.

AWS CLI

Untuk melihat kebijakan berbasis sumber daya dari fungsi, gunakan perintah `get-resource-policy`. Anda dapat menggunakan ARN fungsi yang tidak memenuhi syarat, atau menentukan versi atau alias.

```
aws lambda get-resource-policy --resource-arn arn:aws:lambda:us-
east-2:123456789012:function:my-function
```

Anda juga dapat menggunakan `get-policy` perintah:

```
aws lambda get-policy \
  --function-name my-function \
  --output text
```

Lambda APIs

Untuk mengambil kebijakan berbasis sumber daya fungsi, gunakan tindakan API. GetResourcePolicy Anda dapat menggunakan ARN fungsi yang tidak memenuhi syarat, atau menentukan versi atau alias ARN.

Anda juga dapat menggunakan tindakan GetPolicyAPI.

Menghapus kebijakan berbasis sumber daya fungsi

Console

Untuk menghapus kebijakan berbasis sumber daya fungsi

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih fungsi yang ingin Anda hapus kebijakan izin.
3. Pilih tab Konfigurasi, lalu Izin.
4. Di panel pernyataan Resource-based kebijakan, pilih Hapus.

AWS CLI

Untuk menghapus kebijakan fungsi Anda, gunakan `delete-resource-policy` perintah dan tentukan ARN fungsi, versi, atau alias yang ingin Anda hapus kebijakan tersebut.

```
aws lambda delete-resource-policy --resource-arn arn:aws:lambda:us-
east-2:123456789012:function:my-function
```

Untuk menghapus pernyataan individual, gunakan`remove-permission`.

```
aws lambda remove-permission \
  --function-name example \
  --statement-id sns
```

Lambda APIs

Untuk menghapus kebijakan berbasis sumber daya yang dilampirkan ke fungsi, versi fungsi, atau alias fungsi, gunakan tindakan API. DeleteResourcePolicy

Untuk menghapus pernyataan individual dari kebijakan, gunakan tindakan RemovePermissionAPI.

Memperbarui kebijakan yang ada

Saat memperbarui izin berbasis sumber daya fungsi yang ada, perilakunya bergantung pada metode yang Anda gunakan:

- `put-resource-policy`/PutResourcePolicy— Menggantikan seluruh kebijakan yang ada. Setiap izin individu yang ditambahkan sebelumnya akan ditimpa.

- `add-permission`/AddPermission— Menambahkan pernyataan ke kebijakan yang ada tanpa menimpa. Jika Anda menelepon `add-permission` setelah`put-resource-policy`, pernyataan baru akan ditambahkan ke kebijakan JSON yang ada.

Untuk menghindari penimpaan izin yang ada secara tidak sengaja saat menggunakan`put-resource-policy`, ambil kebijakan fungsi yang ada terlebih dahulu. `get-resource-policy`Outputnya termasuk `RevisionId` bidang.

```
aws lambda get-resource-policy --resource-arn arn:aws:lambda:us-east-2:123456789012:function:my-function
```

Saat Anda melampirkan kebijakan baru, berikan `RevisionId` nilai dengan `--revision-id` parameter untuk memastikan bahwa Anda memperbarui versi terbaru. Jika Anda memberikan ID revisi yang lebih lama, Lambda tidak memperbarui kebijakan fungsi Anda.

```
aws lambda put-resource-policy \
  --resource-arn arn:aws:lambda:us-east-2:123456789012:function:my-function \
  --policy file://policy.json \
  --revision-id a1b2c3d4-5678-90ab-cdef-EXAMPLE11111
```

> **Note**
>
> Kebijakan berbasis sumber daya yang ada dibuat dengan `AddPermission` terus bekerja tanpa modifikasi. Tidak ada perubahan pada kode fungsi Anda yang diperlukan untuk menggunakan `PutResourcePolicy` API baru.

Praktik terbaik keamanan

Dengan kebijakan berbasis sumber daya JSON, Anda dapat mengikuti pola akses hak istimewa paling sedikit. Anda juga dapat memenuhi persyaratan peraturan untuk penolakan eksplisit. Dengan kebijakan JSON lengkap, Anda dapat:

- Buat `Deny` pernyataan eksplisit untuk memblokir prinsip atau kondisi tertentu.
- Gunakan kondisi organisasi (`aws:PrincipalOrgID`,`aws:PrincipalOrgPaths`) untuk membatasi akses ke Organizations Anda tanpa menyebutkan akun individual.
- Izin lingkup ke akun sumber atau ARN tertentu menggunakan rangkaian lengkap kunci kondisi global IAM.

Contoh kebijakan berbasis sumber daya

Example Memberikan izin ke Amazon S3 dengan pernyataan penolakan

Kebijakan berikut memberikan semua bucket Amazon S3 dalam izin akun untuk menjalankan fungsi, kecuali satu bucket yang ditolak secara eksplisit.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "allow-s3",
            "Effect": "Allow",
            "Principal": {
                "Service": "s3.amazonaws.com"
            },
            "Action": "lambda:InvokeFunction",
            "Resource": "arn:aws:lambda:us-east-2:111122223333:function:my-function",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "111122223333"
                }
            }
        },
        {
            "Sid": "deny-s3-bucket",
            "Effect": "Deny",
            "Principal": {
                "Service": "s3.amazonaws.com"
            },
            "Action": "lambda:InvokeFunction",
            "Resource": [
                "arn:aws:lambda:us-east-2:111122223333:function:my-function",
                "arn:aws:lambda:us-east-2:111122223333:function:my-function:*"
            ],
            "Condition": {
                "ArnLike": {
                    "aws:SourceArn": "arn:aws:s3:::amzn-s3-demo-bucket"
                }
            }
        }
    ]
}
```

Example Memberikan izin ke akun dalam suatu organisasi

Hibah kebijakan berikut meminta akses ke semua orang Akun AWS dalam suatu organisasi.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "org-access",
            "Effect": "Allow",
            "Action": "lambda:InvokeFunction",
            "Principal": "*",
            "Resource": "arn:aws:lambda:us-east-2:111122223333:function:my-function",
            "Condition": {
                "StringEquals": {
                    "aws:PrincipalOrgID": "o-a1b2c3d4e5f"
                }
            }
        }
    ]
}
```

Example Memberikan izin untuk beberapa peran IAM dengan kondisi

Kebijakan berikut memberikan izin kepada dua peran IAM untuk menggunakan CreateAlias tindakan dari alamat IP tertentu.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "allow-roles",
            "Effect": "Allow",
            "Action": "lambda:CreateAlias",
            "Principal": {
                "AWS": [
                    "arn:aws:iam::444455556666:role/role-name",
                    "arn:aws:iam::444455556666:role/role-name2"
                ]
            },
            "Resource": "arn:aws:lambda:us-east-2:111122223333:function:my-function",
            "Condition": {
                "IpAddress": {
```

```
                    "aws:SourceIp": "192.0.2.0"
                }
            }
        }
    ]
}
```

Example Menolak akses kecuali penelepon adalah bagian dari organisasi tertentu

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "deny-access",
            "Effect": "Deny",
            "Action": "lambda:InvokeFunction",
            "Principal": "*",
            "Resource": "arn:aws:lambda:us-east-2:111122223333:function:my-function",
            "Condition": {
                "ForAllValues:StringNotLike": {
                    "aws:PrincipalOrgPaths": [
                        "o-a1b2c3d4e5/r-ab12/ou-ab12-11111111/*"
                    ]
                }
            }
        },
        {
            "Sid": "allow-access",
            "Effect": "Allow",
            "Action": "lambda:InvokeFunction",
            "Principal": "*",
            "Resource": "arn:aws:lambda:us-east-2:111122223333:function:my-function",
            "Condition": {
                "ForAnyValue:StringLike": {
                    "aws:PrincipalOrgPaths": [
                        "o-a1b2c3d4e5/r-ab12/ou-ab12-11111111/*"
                    ]
                }
            }
        }
    ]
}
```

Tindakan API yang didukung

Tindakan API Lambda berikut mendukung kebijakan berbasis sumber daya:

- CreateAlias
- DeleteAlias
- DeleteFunction
- DeleteFunctionConcurrency
- DeleteFunctionEventInvokeConfig
- DeleteProvisionedConcurrencyConfig
- GetAlias
- GetFunction
- GetFunctionConcurrency
- GetFunctionConfiguration
- GetFunctionEventInvokeConfig
- GetPolicy
- GetProvisionedConcurrencyConfig
- Memohon
- InvokeFunctionUrl(hanya izin)
- ListAliases
- ListFunctionEventInvokeConfigs
- ListProvisionedConcurrencyConfigs
- ListTags
- ListVersionsByFunction
- PublishVersion
- PutFunctionConcurrency
- PutFunctionEventInvokeConfig
- PutProvisionedConcurrencyConfig
- TagResource
- UntagResource
- UpdateAlias
- UpdateFunctionCode

- UpdateFunctionEventInvokeConfig

Memberikan akses Fungsi Acara ke Layanan AWS

Saat Anda menggunakan AWS layanan untuk menjalankan fungsi, Anda memberikan izin dalam pernyataan tentang kebijakan berbasis sumber daya. Anda dapat menerapkan pernyataan ke seluruh fungsi, atau membatasi pernyataan ke satu versi atau alias.

> **Note**
>
> Saat Anda menambahkan pemicu ke fungsi Anda dengan konsol Lambda, konsol memperbarui kebijakan berbasis sumber daya fungsi untuk memungkinkan layanan memicunya. Untuk memberikan izin ke akun atau layanan lain yang tidak tersedia di konsol Lambda, Anda dapat menggunakan AWS CLI.

Kami merekomendasikan penggunaan `put-resource-policy` untuk menentukan kebijakan JSON lengkap. Dengan`put-resource-policy`, Anda dapat menambahkan pernyataan penolakan, menggunakan berbagai kunci kondisi IAM, dan mengelola semua izin dalam satu dokumen. Anda juga dapat menggunakan `add-permission` untuk menambahkan pernyataan individual untuk kasus penggunaan sederhana.

> **Important**
>
> Menggunakan `put-resource-policy` menggantikan kebijakan berbasis sumber daya yang ada pada sumber daya. Jika sumber daya sudah memiliki izin yang ditentukan`add-permission`, `put-resource-policy` timpa. Gunakan `get-resource-policy` untuk mengambil kebijakan yang ada sebelum membuat perubahan.

Menggunakan kebijakan JSON lengkap (disarankan)

Contoh kebijakan berikut memberikan izin Amazon Simple Notification Service untuk memanggil fungsi bernama`my-function`, tetapi hanya untuk topik tertentu. Ini juga menyangkal doa dari topik tertentu.

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Sid": "allow-sns-my-topic",
            "Effect": "Allow",
            "Principal": {
                "Service": "sns.amazonaws.com"
            },
            "Action": "lambda:InvokeFunction",
            "Resource": "arn:aws:lambda:us-east-2:123456789012:function:my-function",
            "Condition": {
                "ArnLike": {
                    "aws:SourceArn": "arn:aws:sns:us-east-2:123456789012:my-topic"
                }
            }
        },
        {
            "Sid": "deny-sns-other-topic",
            "Effect": "Deny",
            "Principal": {
                "Service": "sns.amazonaws.com"
            },
            "Action": "lambda:InvokeFunction",
            "Resource": [
                "arn:aws:lambda:us-east-2:123456789012:function:my-function",
                "arn:aws:lambda:us-east-2:123456789012:function:my-function:*"
            ],
            "Condition": {
                "ArnLike": {
                    "aws:SourceArn": "arn:aws:sns:us-east-2:123456789012:restricted-
topic"
                }
            }
        }
    ]
}
```

Simpan kebijakan ke file bernama `policy.json` dan terapkan dengan`put-resource-policy`:

```
aws lambda put-resource-policy \
  --resource-arn arn:aws:lambda:us-east-2:123456789012:function:my-function \
  --policy file://policy.json
```

Untuk Amazon S3, sumbernya adalah bucket yang ARN-nya tidak menyertakan ID akun. Mungkin saja Anda bisa menghapus bucket dan akun lain bisa membuat bucket dengan nama yang sama.

Gunakan tombol `aws:SourceAccount` kondisi untuk memastikan bahwa hanya sumber daya di akun Anda yang dapat menjalankan fungsi:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "allow-s3",
            "Effect": "Allow",
            "Principal": {
                "Service": "s3.amazonaws.com"
            },
            "Action": "lambda:InvokeFunction",
            "Resource": "arn:aws:lambda:us-east-2:123456789012:function:my-function",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "123456789012"
                },
                "ArnLike": {
                    "aws:SourceArn": "arn:aws:s3:::my-bucket"
                }
            }
        }
    ]
}
```

Menggunakan izin tambahan

Untuk kasus penggunaan sederhana, Anda dapat menggunakan `add-permission` untuk menambahkan pernyataan individual. Perintah berikut memberikan izin Amazon Simple Notification Service untuk memanggil fungsi bernama: `my-function`

```
aws lambda add-permission \
  --function-name my-function \
  --action lambda:InvokeFunction \
  --statement-id sns \
  --principal sns.amazonaws.com \
  --output text
```

Anda akan melihat output berikut:

```
{"Sid":"sns","Effect":"Allow","Principal":
{"Service":"sns.amazonaws.com"},"Action":"lambda:InvokeFunction","Resource":"arn:aws:lambda:us-
east-2:123456789012:function:my-function"}
```

> **Note**
>
> Jika Anda menelepon `add-permission` setelah`put-resource-policy`, pernyataan baru akan ditambahkan ke kebijakan JSON yang ada.

Untuk membatasi pemanggilan ke sumber daya tertentu, gunakan opsi: `source-arn`

```
aws lambda add-permission \
  --function-name my-function \
  --action lambda:InvokeFunction \
  --statement-id sns-my-topic \
  --principal sns.amazonaws.com \
  --source-arn arn:aws:sns:us-east-2:123456789012:my-topic
```

Untuk Amazon S3, gunakan `source-account` opsi dengan ID akun Anda:

```
aws lambda add-permission \
  --function-name my-function \
  --action lambda:InvokeFunction \
  --statement-id s3-account \
  --principal s3.amazonaws.com \
  --source-arn arn:aws:s3:::amzn-s3-demo-bucket \
  --source-account 123456789012
```

Memberikan akses fungsi ke organisasi

Untuk memberikan izin ke semua akun di AWS Organizationsorganisasi, sebaiknya gunakan `put-resource-policy` untuk membuat kebijakan JSON lengkap dengan kunci `aws:PrincipalOrgID` kondisi.

> **Important**
>
> Menggunakan `put-resource-policy` menggantikan kebijakan berbasis sumber daya yang ada pada sumber daya. Jika sumber daya sudah memiliki izin yang ditentukan`add-`

`permission`, `put-resource-policy` timpa. Gunakan `get-resource-policy` untuk mengambil kebijakan yang ada sebelum membuat perubahan.

Menggunakan kebijakan JSON lengkap (disarankan)

Contoh hibah kebijakan berikut meminta akses ke semua Akun AWS dalam organisasi: `o-a1b2c3d4e5f`

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "org-access",
            "Effect": "Allow",
            "Principal": "*",
            "Action": "lambda:InvokeFunction",
            "Resource": "arn:aws:lambda:us-east-2:123456789012:function:my-function",
            "Condition": {
                "StringEquals": {
                    "aws:PrincipalOrgID": "o-a1b2c3d4e5f"
                }
            }
        }
    ]
}
```

Simpan kebijakan ke file bernama `policy.json` dan terapkan:

```
aws lambda put-resource-policy \
  --resource-arn arn:aws:lambda:us-east-2:123456789012:function:my-function \
  --policy file://policy.json
```

Dengan kebijakan JSON lengkap, Anda juga dapat menolak akses ke akun tertentu dalam suatu organisasi:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "allow-org",
```

```
            "Effect": "Allow",
            "Principal": "*",
            "Action": "lambda:InvokeFunction",
            "Resource": "arn:aws:lambda:us-east-2:123456789012:function:my-function",
            "Condition": {
                "StringEquals": {
                    "aws:PrincipalOrgID": "o-a1b2c3d4e5f"
                }
            }
        },
        {
            "Sid": "deny-specific-account",
            "Effect": "Deny",
            "Principal": {
                "AWS": "arn:aws:iam::999888777666:root"
            },
            "Action": "lambda:InvokeFunction",
            "Resource": [
                "arn:aws:lambda:us-east-2:123456789012:function:my-function",
                "arn:aws:lambda:us-east-2:123456789012:function:my-function:*"
            ]
        }
    ]
}
```

Menggunakan izin tambahan

Untuk kasus penggunaan sederhana, Anda dapat menggunakan `add-permission` dengan `principal-org-id` opsi. Perintah berikut memberikan akses pemanggilan ke semua pengguna dalam organisasi: `o-a1b2c3d4e5f`

```
aws lambda add-permission \
  --function-name my-function \
  --statement-id org-access \
  --action lambda:InvokeFunction \
  --principal * \
  --principal-org-id o-a1b2c3d4e5f
```

> **Note**
>
> Dalam perintah ini, `Principal` adalah`*`. Ini berarti bahwa semua pengguna di organisasi `o-a1b2c3d4e5f` mendapatkan izin pemanggilan fungsi. Jika Anda menentukan peran Akun

AWS atau sebagai`Principal`, maka hanya prinsipal yang mendapatkan izin pemanggilan fungsi, tetapi hanya jika mereka juga merupakan bagian dari organisasi. `o-a1b2c3d4e5f`

Untuk informasi selengkapnya, lihat aws: PrincipalOrg ID di panduan pengguna IAM.

Memberikan akses Fungsi Acara ke akun lain

Untuk berbagi fungsi dengan yang lain Akun AWS, tambahkan pernyataan izin lintas akun ke kebijakan berbasis sumber daya fungsi. Sebaiknya gunakan `put-resource-policy` untuk membuat kebijakan JSON lengkap. Dengan`put-resource-policy`, Anda dapat menentukan beberapa prinsip, menambahkan kondisi, dan membuat pernyataan penolakan. Anda juga dapat menggunakan `add-permission` untuk kasus penggunaan sederhana.

Important

Menggunakan `put-resource-policy` menggantikan kebijakan berbasis sumber daya yang ada pada sumber daya. Jika sumber daya sudah memiliki izin yang ditentukan`add-permission`, `put-resource-policy` timpa. Gunakan `get-resource-policy` untuk mengambil kebijakan yang ada sebelum membuat perubahan.

Menggunakan kebijakan JSON lengkap (disarankan)

Contoh kebijakan berikut memberikan dua Akun AWS izin untuk menjalankan fungsi, dengan satu dibatasi untuk peran IAM tertentu:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "allow-account-invoke",
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "arn:aws:iam::111122223333:root",
                    "arn:aws:iam::444455556666:role/cross-account-role"
                ]
            },
            "Action": "lambda:InvokeFunction",
```

```
            "Resource": "arn:aws:lambda:us-east-2:123456789012:function:my-function"
        }
    ]
}
```

Simpan kebijakan ke file bernama `policy.json` dan terapkan. `resource-arn`Dapat menentukan fungsi, versi, atau alias:

```
aws lambda put-resource-policy \
  --resource-arn arn:aws:lambda:us-east-2:123456789012:function:my-function \
  --policy file://policy.json
```

Anda juga dapat membatasi akses lintas akun ke versi tertentu menggunakan alias. Pemberian kebijakan berikut meminta akses hanya melalui alias: `prod`

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "allow-account-prod-alias",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:root"
            },
            "Action": "lambda:InvokeFunction",
            "Resource": "arn:aws:lambda:us-east-2:123456789012:function:my-
function:prod"
        }
    ]
}
```

Simpan kebijakan ini ke file bernama `alias-policy.json` dan terapkan ke alias:

```
aws lambda put-resource-policy \
  --resource-arn arn:aws:lambda:us-east-2:123456789012:function:my-function:prod \
  --policy file://alias-policy.json
```

Kebijakan berbasis sumber daya memberikan izin bagi akun lain untuk mengakses fungsi, tetapi tidak mengizinkan pengguna dalam akun tersebut untuk melebihi izin mereka. Pengguna di akun lain harus memiliki izin pengguna yang sesuai untuk menggunakan API Lambda.

Anda dapat memberikan akses lintas akun untuk sebagian besar tindakan API yang beroperasi pada fungsi yang sudah ada. Misalnya, Anda dapat memberikan akses ke `lambda:ListAliases` agar akun bisa mendapatkan daftar alias, atau `lambda:GetFunction` agar mengunduh kode fungsi Anda.

Untuk memberi akun lain izin untuk beberapa fungsi, atau untuk tindakan yang tidak beroperasi pada fungsi, kami sarankan Anda menggunakan peran IAM.

Menggunakan izin tambahan

Untuk kasus penggunaan sederhana, Anda dapat menggunakan `add-permission` untuk menambahkan pernyataan individual. Contoh berikut memberikan `111122223333` izin akun untuk memanggil `my-function` dengan alias: `prod`

```
aws lambda add-permission \
  --function-name my-function:prod \
  --statement-id xaccount \
  --action lambda:InvokeFunction \
  --principal 111122223333 \
  --output text
```

Anda akan melihat output berikut:

```
{"Sid":"xaccount","Effect":"Allow","Principal":
{"AWS":"arn:aws:iam::111122223333:root"},"Action":"lambda:InvokeFunction","Resource":"arn:aws:l
east-1:123456789012:function:my-function"}
```

> **Note**
>
> Jika Anda menelepon `add-permission` setelah`put-resource-policy`, pernyataan baru akan ditambahkan ke kebijakan JSON yang ada.

Untuk membatasi akses ke pengguna atau peran di akun lain, tentukan ARN lengkap dari identitas sebagai prinsip. Misalnya, `arn:aws:iam::123456789012:user/developer`.

Memberikan akses lapisan Lambda ke akun lain

Untuk berbagi layer dengan yang lain Akun AWS, tambahkan pernyataan izin lintas akun ke kebijakan berbasis sumber daya lapisan. Jalankan add-layer-version-permissionperintah dan

tentukan ID akun sebagai`principal`. Dalam setiap pernyataan, Anda dapat memberikan izin ke satu akun, semua akun, atau organisasi di AWS Organizations.

Contoh berikut memberikan akun 111122223333 akses ke versi 2 lapisan. `bash-runtime`

```
aws lambda add-layer-version-permission \
  --layer-name bash-runtime \
  --version-number 2 \
  --statement-id xaccount \
  --action lambda:GetLayerVersion \
  --principal 111122223333 \
  --output text
```

Anda akan melihat output yang serupa dengan yang berikut:

```
{"Sid":"xaccount","Effect":"Allow","Principal":
{"AWS":"arn:aws:iam::111122223333:root"},"Action":"lambda:GetLayerVersion","Resource":"arn:aws:
east-1:123456789012:layer:bash-runtime:2"}
```

Izin hanya berlaku untuk satu versi lapisan. Ulangi proses ini tiap kali Anda membuat versi lapisan baru.

Untuk memberikan izin ke semua akun dalam suatu AWS Organizationsorganisasi, gunakan `organization-id` opsi ini. Contoh berikut memberikan semua akun dalam organisasi `o-t194hfs8cz` izin untuk menggunakan versi 3 dari. `my-layer`

```
aws lambda add-layer-version-permission \
  --layer-name my-layer \
  --version-number 3 \
  --statement-id engineering-org \
  --principal '*' \
  --action lambda:GetLayerVersion \
  --organization-id o-t194hfs8cz \
  --output text
```

Anda akan melihat output berikut:

```
{"Sid":"engineering-
org","Effect":"Allow","Principal":"*","Action":"lambda:GetLayerVersion","Resource":"arn:aws:lam
east-2:123456789012:layer:my-layer:3","Condition":{"StringEquals":
{"aws:PrincipalOrgID":"o-t194hfs8cz"}}}"
```

Untuk memberikan izin ke beberapa akun atau organisasi, Anda harus menambahkan beberapa pernyataan.

Menggunakan kontrol akses berbasis atribut di Lambda

Dengan kontrol akses berbasis atribut (ABAC), Anda dapat menggunakan tag untuk mengontrol akses ke sumber daya Lambda Anda. Anda dapat melampirkan tag ke sumber daya Lambda tertentu, melampirkannya ke permintaan API tertentu, atau melampirkannya ke prinsipal AWS Identity and Access Management (IAM) yang membuat permintaan. Untuk informasi selengkapnya tentang cara AWS memberikan akses berbasis atribut, lihat Mengontrol akses ke AWS sumber daya menggunakan tag di Panduan Pengguna IAM.

Anda dapat menggunakan ABAC untuk memberikan hak istimewa paling sedikit tanpa menentukan pola Nama Sumber Daya Amazon (ARN) atau ARN dalam kebijakan IAM. Sebagai gantinya, Anda dapat menentukan tag dalam elemen kondisi kebijakan IAM untuk mengontrol akses. Penskalaan lebih mudah dengan ABAC karena Anda tidak perlu memperbarui kebijakan IAM saat membuat sumber daya baru. Sebagai gantinya, tambahkan tag ke sumber daya baru untuk mengontrol akses.

Di Lambda, tag bekerja pada sumber daya berikut:

- Fungsi—Untuk informasi lebih lanjut tentang fungsi penandaan lihat,. the section called “Tanda”
- Konfigurasi penandatanganan kode—Untuk informasi selengkapnya tentang menandai konfigurasi penandatanganan kode, lihat. the section called “Tag konfigurasi penandatanganan kode”
- Pemetaan sumber acara—Untuk informasi selengkapnya tentang menandai pemetaan sumber peristiwa, lihat. the section called “Tag pemetaan sumber acara”

Tag tidak didukung untuk lapisan.

Anda dapat menggunakan kunci kondisi berikut untuk menulis aturan kebijakan IAM berdasarkan tag:

- aws: ResourceTag/tag-key: Kontrol akses berdasarkan tag yang dilampirkan ke sumber daya Lambda.
- aws: RequestTag/tag-key: Memerlukan tag untuk hadir dalam permintaan, seperti saat membuat fungsi baru.
- aws: PrincipalTag/tag-key: Kontrol apa yang boleh dilakukan oleh kepala sekolah IAM (orang yang membuat permintaan) berdasarkan tag yang dilampirkan ke pengguna atau peran IAM mereka.
- aws: TagKeys: Kontrol apakah kunci tag tertentu dapat digunakan dalam permintaan.

Anda hanya dapat menentukan kondisi untuk tindakan yang mendukungnya. Untuk daftar kondisi yang didukung oleh setiap tindakan Lambda, lihat Kunci tindakan, sumber daya, dan kondisi AWS Lambda di Referensi Otorisasi Layanan. Untuk aws: ResourceTag/tag-key support, lihat “Jenis sumber daya yang ditentukan oleh AWS Lambda.” Untuk TagKeys dukungan aws: RequestTag/tag-key dan aws:, lihat “Tindakan yang ditentukan oleh AWS Lambda.”

Topik

- Amankan fungsi Anda dengan tag

Amankan fungsi Anda dengan tag

Langkah-langkah berikut menunjukkan salah satu cara untuk mengatur izin untuk fungsi menggunakan ABAC. Dalam skenario contoh ini, Anda akan membuat empat kebijakan izin IAM. Kemudian, Anda akan melampirkan kebijakan ini ke peran IAM baru. Terakhir, Anda akan membuat pengguna IAM dan memberikan izin kepada pengguna tersebut untuk mengambil peran baru.

Topik

- Prasyarat
- Langkah 1: Memerlukan tag pada fungsi baru
- Langkah 2: Izinkan tindakan berdasarkan tag yang dilampirkan ke Fungsi Acara dan prinsipal IAM
- Langkah 3: Berikan izin daftar
- Langkah 4: Berikan izin IAM
- Langkah 5: Buat peran IAM
- Langkah 6: Buat pengguna IAM
- Langkah 7: Uji izin
- Langkah 8: Bersihkan sumber daya Anda

Prasyarat

Pastikan Anda memiliki peran eksekusi Lambda. Anda akan menggunakan peran ini saat memberikan izin IAM dan saat Anda membuat Fungsi Acara.

Langkah 1: Memerlukan tag pada fungsi baru

Saat menggunakan ABAC dengan Lambda, ini adalah praktik terbaik untuk mengharuskan semua fungsi memiliki tag. Ini membantu memastikan bahwa kebijakan izin ABAC Anda berfungsi seperti yang diharapkan.

Buat kebijakan IAM yang mirip dengan contoh berikut. Kebijakan ini menggunakan kunci TagKeys kondisi aws: RequestTag/tag-key ResourceTag/tag-key, aws:, dan aws: untuk mengharuskan fungsi baru dan prinsip IAM yang membuat fungsi keduanya memiliki `project` tag. `ForAllValues`Pengubah memastikan bahwa itu `project` adalah satu-satunya tag yang diizinkan. Jika Anda tidak menyertakan `ForAllValues` pengubah, pengguna dapat menambahkan tag lain ke fungsi selama mereka juga lulus`project`.

Example— Memerlukan tag pada fungsi baru

JSON

```
{
    "Version":"2012-10-17",
    "Statement": {
      "Effect": "Allow",
      "Action": [
        "lambda:CreateFunction",
        "lambda:TagResource"
      ],
      "Resource": "arn:aws:lambda:*:*:function:*",
      "Condition": {
        "StringEquals": {
          "aws:RequestTag/project": "${aws:PrincipalTag/project}",
          "aws:ResourceTag/project": "${aws:PrincipalTag/project}"
        },
        "ForAllValues:StringEquals": {
          "aws:TagKeys": "project"
        }
      }
    }
  }
```

Langkah 2: Izinkan tindakan berdasarkan tag yang dilampirkan ke Fungsi Acara dan prinsipal IAM

Buat kebijakan IAM kedua menggunakan aws: ResourceTag/tag-key condition key untuk meminta tag prinsipal agar sesuai dengan tag yang dilampirkan ke fungsi. Contoh kebijakan berikut memungkinkan prinsipal dengan `project` tag untuk memanggil fungsi dengan tag. `project` Jika suatu fungsi memiliki tag lain, tindakan ditolak.

Example— Memerlukan tag yang cocok pada fungsi dan prinsip IAM

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Action": [
          "lambda:InvokeFunction",
          "lambda:GetFunction"
        ],
        "Resource": "arn:aws:lambda:*:*:function:*",
        "Condition": {
          "StringEquals": {
            "aws:ResourceTag/project": "${aws:PrincipalTag/project}"
          }
        }
      }
    ]
  }
```

Langkah 3: Berikan izin daftar

Buat kebijakan yang memungkinkan prinsipal untuk mencantumkan fungsi Lambda dan peran IAM. Hal ini memungkinkan prinsipal untuk melihat semua fungsi Lambda dan peran IAM di konsol dan saat memanggil tindakan API.

Example— Berikan izin daftar Lambda dan IAM

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
      {
        "Sid": "AllResourcesLambdaNoTags",
        "Effect": "Allow",
        "Action": [
          "lambda:GetAccountSettings",
          "lambda:ListFunctions",
          "iam:ListRoles"
        ],
        "Resource": "*"
      }
    ]
  }
```

Langkah 4: Berikan izin IAM

Buat kebijakan yang memungkinkan iam: PassRole. Izin ini diperlukan saat Anda menetapkan peran eksekusi ke suatu fungsi. Dalam contoh kebijakan berikut, ganti contoh ARN dengan ARN peran eksekusi Lambda Anda.

> **Note**
>
> Jangan menggunakan kunci ketentuan `ResourceTag` dalam sebuah kebijakan dengan tindakan `iam:PassRole`. Anda tidak dapat menggunakan tag pada peran IAM untuk mengontrol akses ke siapa yang dapat memberikan peran tersebut. Untuk informasi selengkapnya tentang izin yang diperlukan untuk meneruskan peran ke layanan, lihat Memberikan izin pengguna untuk meneruskan peran ke layanan. AWS

Example— Berikan izin untuk lulus peran eksekusi

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
      {
        "Sid": "VisualEditor0",
        "Effect": "Allow",
        "Action": [
          "iam:PassRole"
        ],
        "Resource": "arn:aws:iam::111122223333:role/lambda-ex"
      }
    ]
  }
```

Langkah 5: Buat peran IAM

Ini adalah praktik terbaik untuk menggunakan peran untuk mendelegasikan izin. Buat peran IAM yang disebut`abac-project-role`:

- Pada Langkah 1: Pilih entitas tepercaya: Pilih AWS akun dan kemudian pilih Akun ini.
- Pada Langkah 2: Tambahkan izin: Lampirkan empat kebijakan IAM yang Anda buat di langkah sebelumnya.
- Pada Langkah 3: Nama, tinjau, dan buat: Pilih Tambahkan tag. Untuk Kunci, masukkan `project`. Jangan masukkan Nilai.

Langkah 6: Buat pengguna IAM

Buat pengguna IAM yang dipanggil`abac-test-user`. Di bagian Setel izin, pilih Lampirkan kebijakan yang ada secara langsung, lalu pilih Buat kebijakan. Masukkan definisi kebijakan berikut. Ganti *111122223333* dengan ID AWS akun Anda. Kebijakan ini memungkinkan `abac-test-user` untuk berasumsi`abac-project-role`.

Example— Izinkan pengguna IAM untuk mengambil peran ABAC

JSON

```
{
    "Version":"2012-10-17",
    "Statement": {
      "Effect": "Allow",
      "Action": "sts:AssumeRole",
      "Resource": "arn:aws:iam::111122223333:role/abac-project-role"
    }
  }
```

Langkah 7: Uji izin

1. Masuk ke AWS konsol sebagai`abac-test-user`. Untuk informasi selengkapnya, lihat Masuk sebagai pengguna IAM.
2. Beralih ke peran `abac-project-role`. Untuk informasi selengkapnya, lihat Beralih ke peran (konsol).
3. Buat fungsi Lambda:
   - Di bawah Izin, pilih Ubah peran eksekusi default, lalu untuk peran Eksekusi, pilih Gunakan peran yang ada. Pilih peran eksekusi yang sama dengan yang Anda gunakanLangkah 4: Berikan izin IAM.
   - Di bawah Pengaturan lanjutan, pilih Aktifkan tag dan kemudian pilih Tambahkan tag baru. Untuk Kunci, masukkan `project`. Jangan masukkan Nilai.
4. Uji fungsinya.
5. Buat fungsi Lambda kedua dan tambahkan tag yang berbeda, seperti. `environment` Operasi ini akan gagal karena kebijakan ABAC yang Anda buat Langkah 1: Memerlukan tag pada fungsi baru hanya memungkinkan prinsipal untuk membuat fungsi dengan `project` tag.
6. Buat fungsi ketiga tanpa tag. Operasi ini akan gagal karena kebijakan ABAC yang Anda buat Langkah 1: Memerlukan tag pada fungsi baru tidak mengizinkan prinsipal untuk membuat fungsi tanpa tag.

Dengan strategi otorisasi ini, Anda dapat mengontrol akses tanpa membuat kebijakan baru untuk setiap pengguna baru. Untuk memberikan akses ke pengguna baru, cukup beri mereka izin untuk mengambil peran yang sesuai dengan proyek yang ditugaskan.

Langkah 8: Bersihkan sumber daya Anda

Untuk menghapus peran IAM

1. Buka halaman Peran dari konsol IAM.
2. Pilih peran yang Anda buat di langkah 5.
3. Pilih Hapus.
4. Untuk mengonfirmasi penghapusan, masukkan nama peran di bidang input teks.
5. Pilih Hapus.

Untuk menghapus pengguna IAM

1. Buka halaman Pengguna konsol IAM.
2. Pilih pengguna IAM yang Anda buat di langkah 6.
3. Pilih Hapus.
4. Untuk mengonfirmasi penghapusan, masukkan nama pengguna di bidang input teks.
5. Pilih Hapus pengguna.

Untuk menghapus Fungsi Acara Lambda

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih Fungsi Acara yang Anda buat.
3. Pilih Tindakan, Hapus.
4. Ketik **`confirm`** kolom input teks dan pilih Hapus.

Fine-tuning Bagian Sumber Daya dan Ketentuan dari kebijakan

Anda dapat membatasi cakupan izin pengguna dengan menentukan sumber daya dan kondisi dalam kebijakan AWS Identity and Access Management (IAM). Setiap tindakan dalam kebijakan mendukung kombinasi sumber daya dan jenis kondisi yang bervariasi tergantung pada perilaku tindakan.

Setiap pernyataan kebijakan IAM memberikan izin untuk tindakan yang dilakukan pada sumber daya. Ketika tindakan tersebut tidak dilakukan berdasarkan sumber daya yang disebutkan, atau ketika Anda memberikan izin untuk melakukan tindakan pada semua sumber daya, nilai sumber daya dalam kebijakan tersebut adalah wildcard (*). Untuk banyak tindakan, Anda dapat membatasi sumber daya yang dapat dimodifikasi pengguna dengan menentukan Nama Sumber Daya Amazon (ARN) sumber daya, atau pola ARN yang cocok dengan beberapa sumber daya.

Berdasarkan jenis sumber daya, desain umum tentang cara membatasi ruang lingkup suatu tindakan adalah sebagai berikut:

- Fungsi — Tindakan yang beroperasi pada suatu fungsi dapat dibatasi pada fungsi tertentu berdasarkan fungsi, versi, atau alias ARN.
- Pemetaan sumber peristiwa — Tindakan dapat dibatasi untuk sumber daya pemetaan sumber peristiwa tertentu oleh ARN. Pemetaan sumber peristiwa selalu dikaitkan dengan suatu fungsi. Anda juga dapat menggunakan `lambda:FunctionArn` kondisi untuk membatasi tindakan dengan fungsi terkait.
- Layer—Tindakan yang terkait dengan penggunaan lapisan dan izin bertindak pada versi lapisan.
- Konfigurasi penandatanganan kode — Tindakan dapat dibatasi untuk sumber daya konfigurasi penandatanganan kode tertentu oleh ARN.
- Tag—Gunakan kondisi tag standar. Untuk informasi selengkapnya, lihat the section called “Attribute-based kontrol akses”.

Untuk membatasi izin berdasarkan sumber daya, tentukan sumber daya berdasarkan ARN.

Format ARN sumber daya Lambda

- Fungsi – `arn:aws:lambda:`*`us-west-2`*`:`*`123456789012`*`:function:`*`my-function`*
- Versi fungsi – `arn:aws:lambda:`*`us-west-2`*`:`*`123456789012`*`:function:`*`my-function`*`:`*`1`*
- Alias fungsi – `arn:aws:lambda:`*`us-west-2`*`:`*`123456789012`*`:function:`*`my-function`*`:`*`TEST`*
- Eksekusi tahan lama — `arn:aws:lambda:`*`us-west-2`*`:`*`123456789012`*`:function:`*`my-function`*`:`*`1`*`/durable-execution/`*`execution-name`*`/`*`execution-id`*
- Pemetaan sumber kejadian – `arn:aws:lambda:`*`us-west-2`*`:`*`123456789012`*`:event-source-mapping:`*`fa123456-14a1-4fd2-9fec-83de64ad683de6d47`*
- Lapisan – `arn:aws:lambda:`*`us-west-2`*`:`*`123456789012`*`:layer:`*`my-layer`*
- Versi lapisan – `arn:aws:lambda:`*`us-west-2`*`:`*`123456789012`*`:layer:`*`my-layer`*`:`*`1`*

- Konfigurasi penandatanganan kode - `arn:aws:lambda:us-west-2:123456789012:code-signing-config:my-csc`

Misalnya, kebijakan berikut memungkinkan pengguna Akun AWS `123456789012` untuk memanggil fungsi bernama `my-function` di Wilayah Barat AS (Oregon) AWS .

Example aktifkan kebijakan fungsi

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "Invoke",
            "Effect": "Allow",
            "Action": [
                "lambda:InvokeFunction"
            ],
            "Resource": "arn:aws:lambda:us-west-2:123456789012:function:my-function"
        }
    ]
}
```

Ini adalah kasus khusus di mana pengidentifikasi tindakan (`lambda:InvokeFunction`) berbeda dari operasi API (Aktifkan). Untuk tindakan lainnya, pengidentifikasi tindakan adalah nama operasi yang diawali dengan `lambda:`.

Bagian-bagian

- Memahami bagian Kondisi dalam kebijakan
- Mereferensikan fungsi di bagian Sumber Daya kebijakan
- Mereferensikan eksekusi yang tahan lama di bagian Sumber Daya kebijakan
- Tindakan IAM dan perilaku fungsi yang didukung

Memahami bagian Kondisi dalam kebijakan

Kondisi adalah elemen kebijakan opsional yang menerapkan logika tambahan untuk menentukan apakah suatu tindakan diperbolehkan. Selain kondisi umum yang didukung semua tindakan, Lambda mendefinisikan jenis kondisi yang dapat Anda gunakan untuk membatasi nilai parameter tambahan pada beberapa tindakan.

Misalnya, `lambda:Principal` kondisi ini memungkinkan Anda membatasi layanan atau akun tempat pengguna dapat memberikan akses pemanggilan pada kebijakan berbasis sumber daya fungsi. Kebijakan berikut memungkinkan pengguna memberikan izin ke topik Amazon Simple Notification Service (Amazon SNS) untuk memanggil fungsi bernama. `test`

Example mengelola izin kebijakan fungsi

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ManageFunctionPolicy",
            "Effect": "Allow",
            "Action": [
                "lambda:AddPermission",
                "lambda:RemovePermission"
            ],
            "Resource": "arn:aws:lambda:us-west-2:123456789012:function:test:*",
            "Condition": {
                "StringEquals": {
                    "lambda:Principal": "sns.amazonaws.com"
                }
            }
        }
    ]
}
```

Kondisi mensyaratkan bahwa prinsipal adalah Amazon SNS dan bukan layanan atau akun lain. Pola sumber daya mengharuskan nama fungsi `test` dan mencakup nomor versi atau alias. Misalnya, `test:v1`.

Untuk informasi selengkapnya tentang sumber daya dan ketentuan untuk Lambda dan AWS layanan lainnya, lihat Kunci tindakan, sumber daya, dan kondisi untuk AWS layanan di Referensi Otorisasi Layanan.

Mereferensikan fungsi di bagian Sumber Daya kebijakan

Anda mereferensikan Fungsi Peristiwa Lambda dalam pernyataan kebijakan menggunakan Amazon Resource Name (ARN). Format fungsi ARN tergantung pada apakah Anda mereferensikan seluruh fungsi (tidak memenuhi syarat) atau versi fungsi atau alias (memenuhi syarat).

Saat melakukan panggilan API Lambda, pengguna dapat menentukan versi atau alias dengan meneruskan versi ARN atau alias ARN dalam parameter, atau dengan menetapkan GetFunction`FunctionName`nilai dalam parameter. GetFunction`Qualifier` Lambda membuat keputusan otorisasi dengan membandingkan elemen sumber daya dalam kebijakan IAM dengan panggilan API `FunctionName` dan yang `Qualifier` diteruskan. Jika ada ketidakcocokan, Lambda menolak permintaan tersebut.

Apakah Anda mengizinkan atau menolak tindakan pada fungsi Anda, Anda harus menggunakan jenis ARN fungsi yang benar dalam pernyataan kebijakan Anda untuk mencapai hasil yang Anda harapkan. Misalnya, jika kebijakan Anda mereferensikan ARN yang tidak memenuhi syarat, Lambda menerima permintaan yang merujuk ARN yang tidak memenuhi syarat tetapi menolak permintaan yang merujuk pada ARN yang memenuhi syarat.

> **Note**
>
> Anda tidak dapat menggunakan karakter wildcard (*) untuk mencocokkan ID akun. Untuk informasi selengkapnya tentang sintaks yang diterima, lihat referensi kebijakan IAM JSON di Panduan Pengguna IAM.

Example memungkinkan pemanggilan ARN yang tidak memenuhi syarat

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "lambda:InvokeFunction",
```

```
            "Resource": "arn:aws:lambda:us-
west-2:123456789012:function:myFunction"
        }
    ]
}
```

Jika kebijakan Anda mereferensikan ARN yang memenuhi syarat tertentu, Lambda menerima permintaan yang mereferensikan ARN tetapi menolak permintaan yang merujuk ARN yang tidak memenuhi syarat atau ARN yang memenuhi syarat lainnya, misalnya,. `myFunction:2`

Example memungkinkan pemanggilan ARN yang memenuhi syarat tertentu

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "lambda:InvokeFunction",
            "Resource": "arn:aws:lambda:us-
west-2:123456789012:function:myFunction:1"
        }
    ]
}
```

Jika kebijakan Anda mereferensikan penggunaan ARN yang memenuhi syarat, `:*` Lambda menerima ARN yang memenuhi syarat tetapi menolak permintaan yang merujuk ARN yang tidak memenuhi syarat.

Example memungkinkan pemanggilan ARN yang memenuhi syarat

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
```

```
            "Action": "lambda:InvokeFunction",
            "Resource": "arn:aws:lambda:us-
west-2:123456789012:function:myFunction:*"
        }
    ]
}
```

Jika kebijakan Anda mereferensikan penggunaan ARN apa pun, * Lambda menerima ARN yang memenuhi syarat atau tidak memenuhi syarat.

Example memungkinkan pemanggilan ARN yang memenuhi syarat atau tidak memenuhi syarat

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "lambda:InvokeFunction",
            "Resource": "arn:aws:lambda:us-
west-2:123456789012:function:myFunction*"
        }
    ]
}
```

Mereferensikan eksekusi yang tahan lama di bagian Sumber Daya kebijakan

Eksekusi yang tahan lama adalah sub-sumber daya dari versi fungsi. Untuk struktur dan pedoman kebijakan ARN eksekusi yang tahan lama, lihat. the section called “Izin IAM untuk fungsi tahan lama”

Anda dapat menjalankan fungsi tahan lama hanya dengan nama fungsi yang memenuhi syarat atau ARN, sehingga kebijakan yang memungkinkan pemanggilan fungsi tahan lama harus sesuai dengan ARN yang memenuhi syarat. Untuk informasi selengkapnya, lihat the section called “Persyaratan ARN yang memenuhi syarat”.

Jika kebijakan Anda mereferensikan fungsi ARN yang tidak memenuhi syarat, Lambda menolak permintaan untuk tindakan eksekusi yang tahan lama. ARN yang tidak memenuhi syarat tidak pernah cocok dengan ARN eksekusi yang tahan lama.

Example Kebijakan yang tidak memberikan akses ke eksekusi yang tahan lama

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "lambda:GetDurableExecution",
            "Resource": "arn:aws:lambda:us-
west-2:123456789012:function:myDurableFunction"
        }
    ]
}
```

Jika kebijakan Anda mereferensikan fungsi ARN dengan, `:*` Lambda menerima permintaan untuk eksekusi tahan lama dari setiap versi.

Example Mengizinkan akses ke eksekusi tahan lama dari versi apa pun

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "lambda:GetDurableExecution",
            "Resource": "arn:aws:lambda:us-
west-2:123456789012:function:myDurableFunction:*"
        }
    ]
}
```

Jika kebijakan Anda mereferensikan fungsi ARN dengan, `:1/durable-execution/*` Lambda menerima permintaan untuk eksekusi versi 1 yang tahan lama saja.

Example Mengizinkan akses ke eksekusi tahan lama dari versi tertentu

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
```

```
            "Action": "lambda:GetDurableExecution",
            "Resource": "arn:aws:lambda:us-
west-2:123456789012:function:myDurableFunction:1/durable-execution/*"
        }
    ]
}
```

⚠ Pola alias tidak pernah cocok dengan ARN eksekusi yang tahan lama

Nama alias tidak pernah muncul di ARN eksekusi yang tahan lama. Pola sumber daya seperti `my-function:prod/durable-execution/*` tidak pernah cocok dengan eksekusi tahan lama karena Lambda menyelesaikan alias ke nomor versi saat fungsi dijalankan.

Lambda mengotorisasi `lambda:ListDurableExecutionsByFunction` tindakan terhadap fungsi ARN dari permintaan, bukan ARN eksekusi yang tahan lama. Aturan berikut berlaku untuk pencocokan ARN:

- Kebijakan yang memberikan fungsi ARN yang tidak memenuhi syarat hanya cocok dengan permintaan yang menggunakan nama fungsi yang tidak memenuhi syarat.
- Kebijakan yang memberikan ARN yang memenuhi syarat hanya cocok dengan permintaan yang menggunakan nama fungsi yang memenuhi syarat.

Example Mengizinkan daftar eksekusi tahan lama dengan salah satu bentuk nama

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "lambda:ListDurableExecutionsByFunction",
            "Resource": [
                "arn:aws:lambda:us-west-2:123456789012:function:myDurableFunction",
                "arn:aws:lambda:us-west-2:123456789012:function:myDurableFunction:*"
            ]
        }
    ]
}
```

Tindakan IAM dan perilaku fungsi yang didukung

Tindakan menentukan apa yang dapat diizinkan melalui kebijakan IAM. Untuk daftar tindakan yang didukung di Lambda, lihat Kunci tindakan, sumber daya, dan kondisi AWS Lambda di Referensi Otorisasi Layanan. Dalam kebanyakan kasus, ketika tindakan IAM mengizinkan tindakan API Lambda, nama tindakan IAM sama dengan nama tindakan API Lambda, dengan pengecualian berikut:

Tindakan API	Tindakan IAM
Memohon	`lambda:InvokeFunction`
GetLayerVersion GetLayerVersionByArn	`lambda:GetLayerVersion`

Selain sumber daya dan kondisi yang ditentukan dalam Referensi Otorisasi Layanan, Lambda mendukung sumber daya dan ketentuan berikut untuk tindakan tertentu. Banyak dari ini terkait dengan fungsi referensi di bagian sumber daya kebijakan. Tindakan yang beroperasi pada fungsi dapat dibatasi pada fungsi tertentu berdasarkan fungsi, versi, atau alias ARN, seperti yang dijelaskan dalam tabel berikut.

Tindakan	Sumber Daya	Kondisi
AddPermission RemovePermission Memohon (Izin:`lambda:InvokeFunction`)	Versi fungsi Alias fungsi	N/A
ListDurableExecutionsByFunction	Versi fungsi	N/A
UpdateFunctionConfiguration	N/A	`lambda:CodeSigningConfigArn`
CreateFunctionUrlConfig DeleteFunctionUrlConfig	Alias fungsi	N/A

Tindakan	Sumber Daya	Kondisi
GetFunctionUrlConfig UpdateFunctionUrlConfig		

AWS Fungsi Web Lambda

Dengan Fungsi AWS Lambda Web, Anda dapat menjalankan aplikasi web dan backend API sebagai server HTTP tanpa server. Kode Anda mendengarkan pada port jaringan dan menangani permintaan secara langsung. Anda dapat menggunakan kerangka kerja Node.js web yang sudah dikenal sepertiExpress,Hono, atau React Router tanpa Lambda-specific adaptor atau pembungkus. Anda mengembangkan dan menjalankan server Anda secara lokal seperti aplikasi Node.js web lainnya.

Anda menerapkan kode Anda sebagai file.zip. Lambda membuat titik akhir HTTPS, mengelola lingkungan eksekusi, dan menskalakan aplikasi Anda dengan lalu lintas, termasuk ke nol saat idle. Untuk mengurangi latensi bagi pengguna yang didistribusikan secara global, Anda dapat menerapkan beberapa di Wilayah AWS belakang satu titik akhir. Lambda menggunakan routing berbasis latensi untuk mengirim setiap permintaan ke salah satu Wilayah tersebut. Anda membayar permintaan, waktu CPU aktif, memori yang dikonsumsi selama masa pakai setiap lingkungan eksekusi, dan transfer data.

Topik

- Kapan menggunakan Fungsi Web
- Cara Kerja Fungsi Web
- Manfaat utama
- Konsep inti
- Membuat Fungsi Web Lambda
- Menerapkan Fungsi Web
- Permintaan dan tanggapan HTTP
- Mengkonfigurasi jaringan untuk AWS Lambda Fungsi Web
- Perilaku penskalaan untuk Lambda Web Functions
- Siklus hidup lingkungan eksekusi
- Runtime Fungsi Web Lambda
- Mengkonfigurasi Fungsi Web Lambda
- Izin Fungsi Web Lambda
- Kontrol akses titik akhir Fungsi Web Lambda
- Enkripsi saat istirahat untuk Fungsi Web
- Memantau Fungsi Web Lambda

- Aplikasi contoh Fungsi Web
- Praktik terbaik untuk Fungsi Web Lambda
- Menguji Fungsi Web Lambda
- Memecahkan Masalah Fungsi Web Lambda

Kapan menggunakan Fungsi Web

Gunakan Fungsi Web untuk beban kerja yang menerima dan merespons lalu lintas HTTP secara langsung. Kasus penggunaan umum meliputi:

- Aplikasi web — Menyebarkan aplikasi full-stack atau front-end yang melayani rute HTML, aset, dan API dari satu penerapan.
- REST dan GraphQL API — Buat backend API yang melayani volume permintaan tinggi.
- Microservices — Jalankan layanan HTTP yang menskalakan secara independen dengan lalu lintas, termasuk ke nol saat idle.
- Server-side rendering (SSR) - Sajikan halaman yang dirender di server pada waktu permintaan daripada pra-dibangun. Fungsi Web menjalankan server produksi kerangka kerja Anda secara langsung. Ini menangani permintaan SSR, streaming, dan rute API tanpa lapisan hosting terpisah. Ini juga berskala dengan lalu lintas. Ini sesuai dengan aplikasi yang membuat konten yang dipersonalisasi atau sering berubah pada setiap permintaan.
- Streaming Respons — Streaming output ke klien saat diproduksi, seperti respons dari model bahasa besar.

Untuk pemrosesan berbasis peristiwa, seperti menangani pesan antrian atau unggahan file, gunakan Lambda Event Functions sebagai gantinya.

> **Note**
>
> AWS Lambda Fungsi Web saat ini hanya mendukung Node.js runtime. Jika Anda menggunakanJava,, Python.NET, atau bahasa lain, lihat Fungsi AWS Lambda Peristiwa, atau the section called “Buat fungsi pertama Anda” MicroVM untuk MicroVM Lambda.

Cara Kerja Fungsi Web

Fungsi Web berisi revisi dan titik akhir. Revisi adalah snapshot yang tidak dapat diubah dari kode dan konfigurasi Anda. Titik akhir merutekan permintaan HTTP ke satu atau dua revisi. Lambda menjalankan server Anda di lingkungan eksekusi yang masing-masing menangani beberapa permintaan bersamaan. Lambda menskalakan lingkungan eksekusi ini dengan lalu lintas. Lihat perinciannya di the section called “Konsep inti”.

Manfaat utama

Fungsi Web memberikan manfaat berikut untuk aplikasi web dan backend API:

- Pengalaman pengembangan web standar — Menyebarkan Node.js aplikasi standar yang mendengarkan pada port. Anda menggunakan kerangka kerja web yang sudah dikenal dan struktur aplikasi normal Anda.
- Penskalaan adaptif - Setiap lingkungan eksekusi melayani beberapa permintaan bersamaan, seperti server web standar. Akibatnya, lingkungan eksekusi yang lebih sedikit menangani lebih banyak lalu lintas. Lingkungan eksekusi yang ada dapat menyerap peningkatan lalu lintas kecil, sehingga lebih sedikit permintaan menunggu lingkungan eksekusi baru dimulai.
- Usage-based penagihan — Anda membayar permintaan, waktu CPU yang digunakan aplikasi dan runtime Anda secara aktif, memori selama masa pakai setiap lingkungan eksekusi, dan transfer data. Permintaan yang menunggu pada kueri database atau panggilan API eksternal menghabiskan sedikit CPU saat menunggu. Namun, memori terus bertambah saat lingkungan eksekusi ada, termasuk waktu idle antar permintaan.
- Streaming respons — Streaming tanggapan ke klien saat aplikasi Anda menghasilkannya, tanpa konfigurasi tambahan.
- Skala ke nol — Lambda menghapus lingkungan eksekusi idle saat tidak ada lalu lintas dan membuat yang baru saat permintaan tiba. Permintaan mungkin menunggu sementara lingkungan eksekusi diinisialisasi. Untuk detail tentang cara kerja penskalaan dan pelambatan, lihat. the section called “Penskalaan”
- Built-in Titik akhir HTTPS - Setiap titik akhir menyediakan domain HTTPS dengan TLS. Anda memilih autentikasi IAM atau otentikasi yang dikelola aplikasi untuk setiap titik akhir. Lihat perinciannya di the section called “Kontrol akses”. Untuk menggunakan domain atau AWS WAF perlindungan khusus, letakkan CloudFront distribusi Amazon di depan titik akhir.
- Endpoint-level kontrol lalu lintas - Tetapkan batas tingkat permintaan (permintaan per detik) pada setiap titik akhir HTTPS, yang diberlakukan Lambda di setiap Wilayah, dan jumlah maksimum

lingkungan eksekusi `maxEnvironments` (), juga diberlakukan di setiap Wilayah. Bersama-sama, ini memungkinkan Anda membatasi lalu lintas yang diterima titik akhir. Mereka juga memungkinkan Anda membatasi berapa banyak beban yang diteruskan titik akhir ke dependensi hilir seperti database dan API eksternal. Lihat informasi yang lebih lengkap di the section called “Penskalaan”.

Konsep inti

Memahami konsep inti berikut membantu Anda menyebarkan dan mengelola aplikasi web secara efektif. Halaman ini mencakup model sumber daya Fungsi Web, status siklus hidup setiap sumber daya, dan cara Lambda menangani permintaan HTTP yang masuk.

Model sumber daya mendefinisikan struktur Fungsi Web. Fungsi Web adalah sumber daya tingkat atas. Ini berisi revisi (snapshot yang tidak dapat diubah dari kode dan konfigurasi Anda) dan titik akhir (titik masuk jaringan yang merutekan lalu lintas HTTP ke revisi Anda).

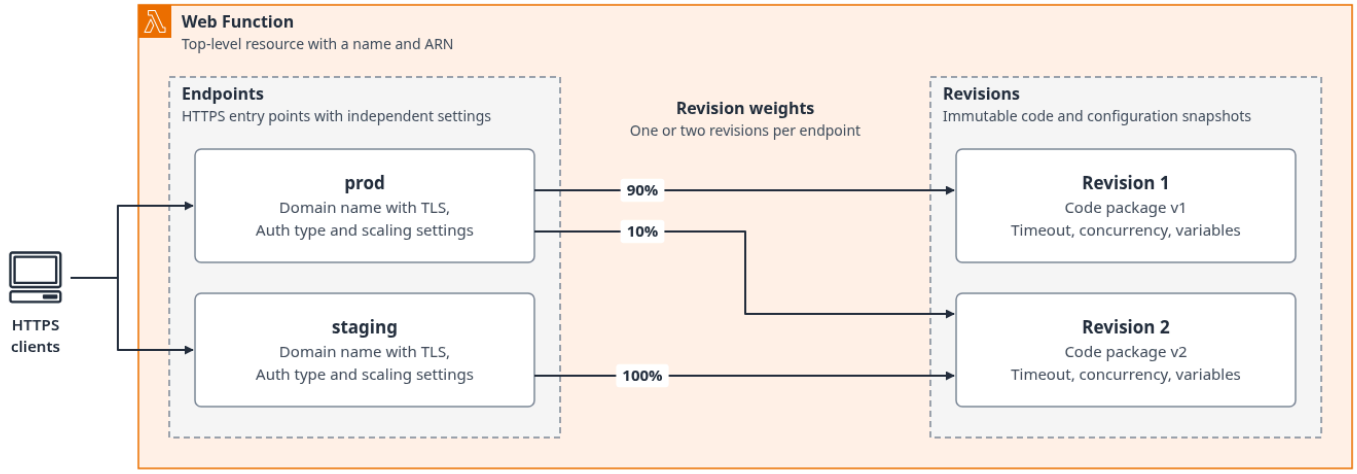
Status revisi dan status titik akhir melacak kemajuan sumber daya tersebut saat Lambda membuat, menyebarkan, dan menghapusnya. Memantau status ini memberi tahu Anda kapan Lambda selesai membuat, menerapkan, atau menghapus sumber daya dan membantu Anda mendiagnosis kegagalan.

Lingkungan eksekusi adalah contoh komputasi terisolasi di mana server web dan aplikasi Anda berjalan. Setiap lingkungan eksekusi menangani beberapa permintaan HTTP bersamaan. Lambda membuat dan menghapus lingkungan secara otomatis berdasarkan konfigurasi dan lalu lintas Fungsi Web Anda.

Bagian berikut memberikan informasi rinci tentang setiap konsep inti.

Model sumber daya

Fungsi Web Lambda terdiri dari tiga jenis sumber daya dalam hierarki orangtua-anak. Fungsi Web adalah sumber daya tingkat atas. Revisi dan endpoint adalah sumber daya turunan dari Fungsi Web. Menyebarkan kode atau mengubah konfigurasi revisi membuat revisi baru. Diagram berikut menunjukkan Fungsi Web yang titik akhirnya mengarahkan lalu lintas ke revisinya.



Fungsi Web

Sumber daya tingkat atas yang mewakili aplikasi web Anda. Fungsi Web memiliki nama, ARN, dan status siklus hidupnya sendiri (Tertunda, Aktif, Gagal, atau Menghapus). Ini dapat berisi revisi dan titik akhir; semua revisi dan titik akhir milik Fungsi Web induk tunggal.

Revisi

Cuplikan yang tidak dapat diubah dari kode dan konfigurasi aplikasi Anda. Anda membuat revisi baru setiap kali mengubah kode atau konfigurasi revisi, seperti batas waktu permintaan, jumlah maksimum permintaan bersamaan per lingkungan eksekusi, atau variabel lingkungan. Karena revisi tidak dapat diubah, Anda selalu dapat memutar kembali ke versi aplikasi sebelumnya dengan merutekan lalu lintas ke revisi sebelumnya. Anda dapat mendistribusikan lalu lintas antara dua revisi sekaligus untuk penerapan kenari atau biru-hijau.

Titik akhir

Titik masuk jaringan yang merutekan lalu lintas HTTP ke revisi Anda. Setiap titik akhir memiliki nama domain unik dengan TLS, jenis otentikasi yang mengontrol akses, dan pengaturan penskalaan yang menentukan kapasitas. Saat Anda membuat titik akhir, Anda memilih jenis titik akhir, yang menetapkan cakupan geografisnya. HomeRegionmelayani lalu lintas dari daerah asal saja. MultiRegionmelayani lalu lintas dari beberapa Wilayah di belakang satu domain global, yang merutekan setiap permintaan ke Wilayah berdasarkan latensi jaringan, dengan upaya terbaik. PerRegionmelayani lalu lintas dari beberapa Wilayah, masing-masing dengan domain Regionalnya sendiri, dan Anda mengontrol bagaimana klien mencapai setiap Wilayah. Fungsi Web dapat memiliki beberapa titik akhir, masing-masing dengan konfigurasi independen. Anda mengonfigurasi bobot revisi pada titik akhir untuk merutekan lalu lintas ke satu atau dua revisi aktif.

Status revisi

Revisi bergerak melalui status berikut selama siklus hidupnya. Anda dapat memeriksa status saat ini dengan menelepon `GetWebFunctionRevision` atau melihat detail revisi di konsol Lambda.

Status	Deskripsi	Apa yang terjadi selanjutnya
Tertunda	Lambda sedang membuat revisi. Ini termasuk mengunduh kode Anda dari Amazon S3 dan memvalidasi konfigurasi.	Revisi bertransisi ke Aktif saat pembuatan berhasil, atau ke Gagal jika terjadi kesalahan.
Aktif	Revisi berhasil dibuat dan siap diletakkan di belakang titik akhir.	Revisi tetap Aktif sampai Anda menghapusnya, Lambda menghapus nya selama pembersihan revisi otomatis (the section called “Batas revisi dan pembersihan otomatis” lihat), atau Anda menghapus akses ke kode sumber. Jika Lambda kehilangan akses ke kode, misalnya karena versi objek Amazon S3 yang direferensikan telah dihapus, revisi akan beralih ke Gagal (lihat). the section called “Siklus hidup objek”
Gagal	Lambda tidak dapat membuat revisi. Masalah yang terdeteksi Lambda saat Anda mengirimkan permintaa n tidak menghasilkan revisi Gagal. Misalnya, jika AWS KMS kunci memiliki izin yang tidak mencukupi, permintaan gagal dengan pengecual ian klien, dan Lambda tidak membuat revisi.	Periksa `stateReason` bidang dalam `GetWebFunctionRevi sion` respons untuk detailnya. Buat revisi baru untuk memperbai ki masalah. Revisi yang gagal tidak dapat melayani lalu lintas.

Status titik akhir

Titik akhir memiliki dua bidang status: `state` melacak siklus hidup keseluruhan, dan `updateStatus` melacak pembaruan titik akhir setelah pembuatan awal. Anda dapat memeriksa keduanya dengan meneleponGetWebFunctionEndpoint.

Status	Deskripsi	Apa yang terjadi selanjutnya
Tertunda	Lambda menciptakan titik akhir. Ini termasuk mengalokasikan nama domain, penyediaan TLS, dan mengonfigurasi perutean. Untuk `MultiRegion` dan `PerRegion` titik akhir, ini juga termasuk mereplikasi konfigurasi ke yang lain. Wilayah AWS	Titik akhir bertransisi ke Aktif saat pembuatan berhasil di semua Wilayah target, atau ke Gagal jika terjadi kesalahan.
Aktif	Endpoint siap menerima lalu lintas HTTP pada nama domainnya. Klien dapat mengirim permintaan ke titik akhir.	Titik akhir tetap aktif. Lambda melacak pembaruan titik akhir berikutnya di `updateStatus` lapangan tanpa mengubah status keseluruhan titik akhir.
Gagal	Penciptaan gagal. Untuk `MultiRegion` dan `PerRegion` titik akhir, ini dapat menunjukkan kegagalan di satu atau lebih Wilayah target.	Periksa `stateReason` bidang untuk detailnya. Untuk `MultiRegion` dan `PerRegion` titik akhir, periksa `regionalEndpoints` entri untuk `state` dan setiap `stateReason` Wilayah.
Deleting	Lambda menghapus titik akhir dan menghentikan nama domainnya. Untuk `MultiRegion` dan `PerRegion` titik akhir, Lambda menghapus titik akhir dari semua Wilayah.	Setelah penghapusan selesai, titik akhir dihapus secara permanen. Nama domainnya sudah pensiun, dan Lambda tidak pernah menetapkannya ke titik akhir lain di akun mana pun. Operasi ini tidak dapat dibalik.

Untuk titik akhir yang ada, Lambda `state` terus disetel selama `Active` pembaruan titik akhir. Periksa `updateStatus` untuk menentukan apakah pembaruan terbaru adalah`InProgress`,`Successful`, atau`Failed`. Misalnya, saat Anda mengubah bobot revisi titik akhir, `updateStatus` adalah `InProgress` saat tetap. `state` `Active` Saat bobot baru berlaku, `updateStatus` ubah menjadi`Successful`.

InProgress

Lambda menerapkan pembaruan titik akhir.

Sukses

Pembaruan endpoint terbaru selesai dan berlaku.

Gagal

Pembaruan endpoint terbaru gagal. Periksa `updateStatusReason` bidang dan `regionalEndpoints` untuk detailnya. Untuk `MultiRegion` dan `PerRegion` titik akhir, beberapa Wilayah mungkin menggunakan konfigurasi sebelumnya sementara Wilayah lain menggunakan konfigurasi yang diminta. Perbaiki penyebab yang dilaporkan, lalu panggil `UpdateWebFunctionEndpoint` lagi dengan konfigurasi yang dimaksudkan untuk merekonsiliasi titik akhir.

Status Fungsi Web, status revisi, dan status titik akhir bersifat independen. Fungsi Web dapat berada dalam status Aktif sementara salah satu titik akhirnya masih Tertunda atau Gagal. Demikian pula, revisi dapat Aktif bahkan jika tidak ada titik akhir yang saat ini merutekan lalu lintas ke sana.

Revisi Aktif, titik akhir Aktif, atau pembaruan titik akhir yang berhasil tidak mengonfirmasi bahwa server Anda dimulai. Lambda tidak memeriksa apakah server Anda mendengarkan saat membuat atau memperbarui titik akhir untuk menggunakan revisi. Setelah setiap penerapan, kirim permintaan pengujian dan periksa log aplikasi Anda.

Lingkungan eksekusi

Lingkungan eksekusi adalah instance komputasi terisolasi tempat server web dan aplikasi Anda berjalan. Saat lalu lintas tiba di titik akhir Anda, Lambda merutekan permintaan ke lingkungan eksekusi yang menjalankan kode revisi aktif.

Lambda mengelola lingkungan eksekusi secara otomatis:

1. Scale up — Ketika lalu lintas meningkat, Lambda menciptakan lingkungan eksekusi tambahan dan memulai server HTTP Anda di masing-masing. Lambda meningkatkan lingkungan baru sebelum permintaan untuk menjaga ruang kepala, jadi permintaan jarang menunggu lingkungan baru dimulai (awal yang dingin). Ketika Fungsi Web menskalakan dari nol (permintaan pertama setelah periode tidak ada lalu lintas), ada penundaan awal dingin singkat sementara Lambda menciptakan lingkungan eksekusi. Permintaan menunggu sementara lingkungan eksekusi diinisialisasi. Permintaan mungkin masih gagal jika aplikasi Anda tidak dimulai atau jika tingkat permintaan, lingkungan eksekusi, atau batas akun berlaku. Lihat perinciannya di the section called “Penskalaan”.
2. Scale down — Ketika lalu lintas menurun, Lambda menghapus lingkungan eksekusi yang tidak lagi dibutuhkan, setelah permintaan dalam penerbangan selesai dengan upaya terbaik. Lambda menskalakan ke lingkungan nol ketika tidak ada lalu lintas.

Jumlah lingkungan eksekusi dan konkurensi mereka dikendalikan oleh dua pengaturan:

- **`maxEnvironments`**(pengaturan titik akhir) — Jumlah maksimum lingkungan eksekusi yang dapat dibuat Lambda untuk titik akhir di setiap Wilayah. Pengaturan ini bersifat opsional. Jika tidak disetel, Lambda menskalakan lingkungan eksekusi sesuai kebutuhan, tunduk pada kuota akun Anda.
- **`maxConcurrencyPerEnvironment`**(pengaturan revisi) — Jumlah maksimum permintaan HTTP yang ditangani oleh satu lingkungan eksekusi pada saat yang bersamaan. Nilai yang valid adalah 1-128. Defaultnya adalah 64.

Lambda menetapkan batas konkurensi dinamis untuk setiap lingkungan eksekusi. Batas ini tidak pernah melebihi`maxConcurrencyPerEnvironment`. Batas atas untuk konkurensi titik akhir `maxEnvironments` dikalikan dengan batas dinamis ini. Kapasitas aktual mungkin lebih rendah karena Lambda menyesuaikan konkurensi per lingkungan berdasarkan pemanfaatan sumber daya dan menerapkan titik akhir dan batas akun. Misalnya, Anda mengatur `maxEnvironments` ke 20 dan `maxConcurrencyPerEnvironment` 64. Titik akhir kemudian menangani hingga 1.280 permintaan bersamaan di Wilayah. Ini menangani lebih sedikit jika Lambda menetapkan batas dinamis di bawah 64. Untuk panduan tentang pengaturan ini, lihatthe section called “Penskalaan”.

Permintaan penanganan

Kode Fungsi Web Anda berjalan sebagai server HTTP standar yang mendengarkan pada port. Lambda mengirimkan permintaan HTTP masuk ke server Anda dan mengembalikan tanggapan ke klien. Anda dapat menggunakan kerangka Node.js web, middleware, atau router apa pun.

Siklus hidup permintaan berfungsi sebagai berikut:

1. Klien mengirimkan permintaan HTTP ke nama domain endpoint.
2. Lambda mengautentikasi permintaan (untuk IamAuthtitik akhir) atau meneruskannya (untuk titik akhir). ApplicationManaged
3. Lambda merutekan permintaan ke lingkungan eksekusi berdasarkan bobot revisi titik akhir.
4. Server HTTP lingkungan eksekusi memproses permintaan dan mengembalikan respons.
5. Lambda mengembalikan respons ke klien.

Setiap permintaan dapat berjalan hingga 900 detik (15 menit). Anda dapat mengonfigurasi batas waktu dari 3 hingga 900 detik menggunakan pengaturan revisi. `timeoutSeconds` Waktu habis default adalah 30 detik.

Setiap lingkungan eksekusi Web Function menangani beberapa permintaan bersamaan, sehingga kode aplikasi Anda harus aman untuk digunakan bersamaan. Untuk detail tentang model konkurensi, lihatthe section called “Node.js model eksekusi”.

Mengelola revisi Fungsi Web

Revisi adalah snapshot yang tidak dapat diubah dari kode dan konfigurasi aplikasi Web Function Anda. Untuk mengubah kode Anda atau pengaturan revisi apa pun, sepertitimeoutSeconds,maxConcurrencyPerEnvironment, atauenvironmentVariables, Anda membuat revisi baru. Apakah revisi baru menerima lalu lintas tergantung pada mode penyebaran otomatis setiap titik akhir. Titik akhir dalam LatestRevisionmode merutekan semua lalu lintas ke revisi baru setelah menjadi Aktif. Titik akhir dalam mode Dinonaktifkan mempertahankan bobot revisi saat ini hingga Anda mengubahnya. Karena revisi tidak dapat diubah, Anda dapat memutar kembali dengan merutekan lalu lintas ke revisi sebelumnya. Revisi sebelumnya harus tetap Aktif, dan kode, kunci, peran, dan dependensi lainnya harus tetap tersedia. Untuk informasi tentang perutean lalu lintas antar revisi, lihat. the section called “Pola deployment”

Properti revisi

Setiap revisi berisi konfigurasi berikut:

Properti	Deskripsi	Default	Dapat diubah
RevisionID	ID unik yang dihasilkan Lambda saat Anda membuat revisi.	–	Tidak (dihasilkan)
RevisionARN	Nama Sumber Daya Amazon (ARN) dari revisi.	–	Tidak (dihasilkan)
FungsiARN	ARN dari Fungsi Web yang dimiliki revisi.	–	Tidak (dihasilkan)
negara	Status siklus hidup revisi (`Pending`,`Active`, atau). `Failed` Lihat perinciannya di the section called “Status revisi”.	–	Tidak (dikelola oleh Lambda)
StateReason	Deskripsi mengapa revisi dalam keadaan saat ini, misalnya penyebab suatu `Failed` negara.	–	Tidak (dikelola oleh Lambda)
kesalahan	Daftar kesalahan yang ditemukan Lambda saat membuat atau membaca revisi. Setiap kesalahan memberi nama atribut yang terpengaruh dan menyertakan kode kesalahan dan pesan. Absen ketika revisi tidak memiliki kesalahan.	–	Tidak (dikelola oleh Lambda)
createDat	Stempel waktu saat revisi dibuat.	–	Tidak (dihasilkan)
deskripsi	Deskripsi opsional untuk revisi yang Anda berikan saat Anda membuatnya.	Tidak ada	Tidak (tidak

Properti	Deskripsi	Default	Dapat diubah
			dapat diubah)
BuildConfig	Kode dan runtime untuk revisi. Berisi `codeConfig` dan `runtimeConfig` pengaturan yang dijelaskan dalam baris berikut.	–	Tidak (tidak dapat diubah)
CodeConfig	Lokasi paket penyebaran. Berisi `s3Object` pengaturan yang dijelaskan dalam baris berikut. Bagian dari `buildConfig` .	–	Tidak (tidak dapat diubah)
Objek S3	Lokasi Amazon S3 dari paket penyebaran. Berisi `bucket`, `key`, dan `versionId` pengaturan yang dijelaskan dalam baris berikut. Lihat perinciannya di the section called “Kode di Amazon S3”. Bagian dari `codeConfig` .	–	Tidak (tidak dapat diubah)
bucket	Nama bucket Amazon S3 yang berisi paket penerapan. Bucket harus mengaktifkan versi S3. Bagian dari `s3Object`.	–	Tidak (tidak dapat diubah)
kunci	Kunci objek Amazon S3 dari paket penerapan. Bagian dari `s3Object`.	–	Tidak (tidak dapat diubah)

Properti	Deskripsi	Default	Dapat diubah
VersionID	ID versi objek Amazon S3. Tidak wajib. Jika Anda menghilangkannya, Lambda menggunakan versi terbaru dari objek pada waktu pembuatan. Either way, Lambda menyematkan revisi ke versi objek tertentu. Bagian dari`s3Object`.	Versi terbaru	Tidak (tidak dapat diubah)
RuntimeConfig	Konfigurasi runtime. Berisi `runtime` pengaturan yang dijelaskan di baris berikut. Bagian dari`buildConfig` .	–	Tidak (tidak dapat diubah)
runtime	Pengidentifikasi runtime (misalnya ,`nodejs24.x`). Untuk runtime yang didukung, lihatthe section called “Runtime”. Meskipun pengidentifikasi runtime diperbaiki untuk revisi, runtime itu sendiri tidak dibekukan. Lambda secara otomatis menerapkan patch keamanan dan perbaikan bug ke runtime, sehingga pengenal (misalnya ,`nodejs24.x`) selalu menyelesa ikan ke versi patch terbaru. Lihat perinciannya di the section called “Penambalan runtime otomatis”. Bagian dari`runtimeConfig` .	–	Tidak (tidak dapat diubah)

Properti	Deskripsi	Default	Dapat diubah
ServiceConfig	Pengaturan runtime untuk revisi. Berisi`executionRoleArn` ,`timeoutSeconds` ,`maxConcurrencyPerEnvironment` ,`environmentVariables` , dan `telemetryConfig` pengaturan yang dijelaskan dalam baris berikut.	–	Tidak (tidak dapat diubah)
eksekusi RoleArn	ARN dari peran IAM yang diasumsikan oleh Fungsi Web saat runtime. Peran tersebut memberikan izin untuk mengakses AWS layanan dan sumber daya. Wajib. Lihat perinciannya di the section called “Peran eksekusi”. Bagian dari`serviceConfig` .	–	Tidak (tidak dapat diubah)
TimeoutSeconds	Waktu maksimum dalam hitungan detik yang dapat dijalankan oleh satu permintaan HTTP. Nilai yang valid: 3—900. Bagian dari`serviceConfig` .	30	Tidak (tidak dapat diubah)
maks ConcurrencyPerEnvironment	Jumlah maksimum permintaan HTTP bersamaan per lingkungan eksekusi. Nilai yang valid: 1—128. Bagian dari`serviceConfig` .	64	Tidak (tidak dapat diubah)
Variabel LingkungannVariabel	Key-value pasangan tersedia untuk aplikasi Anda saat runtime. Tetapkan per-revisi dan abadi setelah pembuatan. Bagian dari`serviceConfig` .	Tidak ada	Tidak (tidak dapat diubah)

Properti	Deskripsi	Default	Dapat diubah
TelemetryConfig	Pengaturan telemetri untuk revisi. Berisi `loggingConfig` pengaturan yang dijelaskan dalam baris berikut. Jika Anda menghilangkan `telemetryConfig`, Lambda masih mengirimkan log aplikasi Anda ke grup log default. Bagian dari `serviceConfig`.	Tidak ada	Tidak (tidak dapat diubah)
LoggingConfig	Konfigurasi logging untuk revisi. Berisi `logGroup`, `applicationLogLevel`, dan `systemLogLevel` pengaturan yang dijelaskan dalam baris berikut. Lihat perinciannya di the section called “Kontrol logging”. Bagian dari `telemetryConfig`.	Tidak ada	Tidak (tidak dapat diubah)
LogGroup	Grup CloudWatch log Amazon Logs yang menerima log Anda. Nama grup log yang dimulai dengan `aws/` dicadangkan dan ditolak. Lihat perinciannya di the section called “Menggunakan grup log kustom”. Bagian dari `loggingConfig`.	`/aws/lambda/web/`*`functionname`*	Tidak (tidak dapat diubah)
aplikasi LogLevel	Tingkat keparahan minimum log aplikasi Anda yang dikirim Lambda ke Amazon CloudWatch Logs. Nilai yang valid: `TRACE`, `DEBUG`, `INFO`, `WARN`, `ERROR`, dan `FATAL`. Bagian dari `loggingConfig`.	`INFO`	Tidak (tidak dapat diubah)

Properti	Deskripsi	Default	Dapat diubah
sistem LogLevel	Tingkat keparahan minimum log Lambda-generated sistem yang dikirim Lambda ke Amazon CloudWatch Logs. Nilai-nilai yang valid: `DEBUG`, `INFO`, dan `WARN`. Bagian dari`loggingConfig` .	`INFO`	Tidak (tidak dapat diubah)
km KeyArn	ARN dari kunci yang dikelola pelanggan digunakan untuk mengenkripsi kode dan variabel lingkungan. Tidak wajib. Jika suatu `MultiRegion` atau `PerRegion` titik akhir merutekan lalu lintas ke revisi, gunakan kunci Multi-region dengan replika di setiap Wilayah titik akhir. Lihat perinciannya di the section called “Enkripsi saat diam”.	Tidak ada	Tidak (tidak dapat diubah)

Semua properti revisi tidak dapat diubah setelah pembuatan. Untuk mengubah konfigurasi apa pun, buat revisi baru dengan nilai yang diperbarui.

Membuat revisi

Lambda membuat revisi saat Anda menelepon. `CreateWebFunctionRevision` Anda memberikan lokasi kode dan konfigurasi dalam permintaan. Revisi baru tidak menyalin pengaturan dari revisi yang ada. Tentukan setiap pengaturan yang ingin Anda pertahankan, termasuk variabel lingkungan. Pengaturan yang Anda hilangkan menggunakan defaultnya. Untuk memulai dari nilai saat ini, jalankan `get-web-function-revision` dan gunakan kembali `buildConfig` dan `serviceConfig` dengan perubahan Anda. Revisi baru dimulai dalam status Pending sementara Lambda mengunduh kode Anda dan menyiapkan artefak kode darinya. Ketika persiapan selesai, revisi beralih ke Aktif. Jika terjadi kesalahan selama persiapan, revisi bertransisi ke Gagal dan tidak dapat melayani lalu lintas. Untuk detail tentang setiap negara bagian, lihatthe section called “Status revisi”.

Sebelum membuat revisi, unggah paket deployment .zip Anda ke bucket Amazon S3 berversi, dan dapatkan ARN dari peran eksekusi yang dapat diasumsikan oleh Lambda. Untuk detailnya, lihat the section called “Kode di Amazon S3” dan the section called “Peran eksekusi”.

Untuk membuat revisi dengan kode yang diperbarui (AWS CLI)

1. Jalankan perintah berikut untuk membuat revisi baru. Ganti bucket S3, key, role ARN, dan nilai konfigurasi dengan milik Anda sendiri.

```
aws lambda-web create-web-function-revision \
    --function-name my-web-app \
    --build-config '{
      "codeConfig": {
        "s3Object": {
          "bucket": "my-bucket",
          "key": "my-app-v2.zip"
        }
      },
      "runtimeConfig": { "runtime": "nodejs24.x" }
    }' \
    --service-config '{
      "executionRoleArn": "arn:aws:iam::111122223333:role/my-web-function-role"
    }'
```

   Perintah menggunakan objek berikut. Untuk setiap pengaturan dan defaultnya, lihatthe section called “Properti revisi”.

   - `buildConfig`berisi`codeConfig`, bucket Amazon S3 dan kunci paket penerapan Anda, dan`runtimeConfig`, runtime (misalnya,). `nodejs24.x` Keduanya diperlukan.
   - `serviceConfig`berisi ARN `executionRoleArn` yang diperlukan dari peran eksekusi yang diasumsikan oleh Fungsi Web Anda saat dijalankan. Itu juga bisa berisi`timeoutSeconds`,`maxConcurrencyPerEnvironment`,`environmentVariables`, dan`telemetryConfig`.

   Perintah mengembalikan rincian tentang revisi baru, termasuk. `revisionId` Revisi dimulai dalam status Tertunda.

2. Untuk mengonfirmasi bahwa revisi Aktif, jalankan perintah berikut. Ganti *my-web-app* dengan nama fungsi Anda dan *rev-a1b2c* dengan `revisionId` dari respons.

```
aws lambda-web get-web-function-revision \
    --function-name my-web-app \
    --revision-id rev-a1b2c
```

Revisi sudah siap kapan`state`. `Active` Jika `state` ya`Failed`, `stateReason` jelaskan penyebabnya. Lihat perinciannya di the section called “Status revisi”.

Tepat setelah Anda membuat revisi, `state` biasanya`Pending`. Untuk menunggu sampai selesai`Active`, jalankan `aws lambda-web wait web-function-revision-active` dengan opsi yang sama.

Untuk membuat revisi dengan konfigurasi yang diperbarui (AWS CLI)

1. Untuk mengubah konfigurasi tanpa mengubah kode, tentukan lokasi kode yang sama dengan pengaturan baru. Contoh berikut mengubah batas waktu menjadi 60 detik dan menambahkan variabel lingkungan.

```
aws lambda-web create-web-function-revision \
    --function-name my-web-app \
    --build-config '{
      "codeConfig": {
        "s3Object": {
          "bucket": "my-bucket",
          "key": "my-app-v2.zip"
        }
      },
      "runtimeConfig": { "runtime": "nodejs24.x" }
    }' \
    --service-config '{
      "executionRoleArn": "arn:aws:iam::111122223333:role/my-web-function-role",
      "timeoutSeconds": 60,
      "maxConcurrencyPerEnvironment": 32,
      "environmentVariables": {
        "DATABASE_URL": "my-database-endpoint",
        "LOG_LEVEL": "info"
      }
    }'
```

2. Untuk mengonfirmasi bahwa revisi Aktif, jalankan perintah berikut. Ganti *my-web-app* dengan nama fungsi Anda dan *rev-a1b2c* dengan `revisionId` dari respons.

```
aws lambda-web get-web-function-revision \
    --function-name my-web-app \
    --revision-id rev-a1b2c
```

Revisi sudah siap kapan`state`. `Active` Jika `state` ya`Failed`, `stateReason` jelaskan penyebabnya. Lihat perinciannya di the section called “Status revisi”.

Tepat setelah Anda membuat revisi, `state` biasanya`Pending`. Untuk menunggu sampai selesai`Active`, jalankan `aws lambda-web wait web-function-revision-active` dengan opsi yang sama.

Setelah membuat revisi, bagaimana lalu lintas mencapainya tergantung pada mode penyebaran otomatis titik akhir. Lihat perinciannya di the section called “Pola deployment”.

Revisi daftar

Untuk melihat semua revisi untuk Fungsi Web, gunakan perintah berikut:

```
aws lambda-web list-web-function-revisions \
    --function-name my-web-app
```

Responsnya mencakup pengidentifikasi, status, stempel waktu pembuatan, dan ringkasan konfigurasi setiap revisi. Gunakan pengenal revisi saat memperbarui bobot revisi titik akhir atau saat menghapus revisi.

Menghapus revisi

Anda dapat menghapus revisi yang tidak lagi diperlukan. Menghapus revisi secara permanen menghapusnya. Paket penerapan Anda di Amazon S3 tidak terpengaruh.

> **Important**
>
> Anda tidak dapat menghapus revisi yang mengarahkan lalu lintas titik akhir mana pun (revisi dengan bobot bukan nol pada titik akhir apa pun). Hapus revisi dari bobot revisi setiap titik akhir sebelum menghapusnya.

Untuk menghapus revisi (AWS CLI)

- Jalankan perintah berikut untuk menghapus revisi tertentu.

```
aws lambda-web delete-web-function-revision \
    --function-name my-web-app \
    --revision-id rev-a1b2c
```

Batas revisi dan pembersihan otomatis

Lambda menggunakan 50 revisi per Fungsi Web sebagai target pembersihan otomatisnya. Targetnya tidak bisa disesuaikan. Di konsol Service Quotas, muncul sebagai Revisi per Fungsi Web (). `web-function-revisions-per-function`

Anda tidak perlu menghapus revisi lama untuk tetap berada dalam target pembersihan. Ketika revisi baru menjadi Aktif dan jumlah revisi total melebihi target pembersihan, Lambda secara otomatis menghapus revisi tertua yang tidak digunakan. Revisi sedang digunakan ketika titik akhir merutekan lalu lintas ke sana, termasuk ketika direferensikan di mana saja di `regionalEndpoints` properti titik akhir atau titik akhir. `MultiRegion` `PerRegion` Lambda tidak secara otomatis menghapus revisi yang sedang digunakan atau revisi yang baru saja menjadi Aktif. Pembersihan tidak sinkron. Jumlah revisi mungkin untuk sementara melebihi 50 sementara revisi yang lebih lama masih digunakan atau pembersihan sedang berlangsung. Revisi yang Anda hapus dari semua bobot revisi, seperti target rollback, dapat dihapus dengan pembersihan.

Anda juga dapat menghapus revisi secara manual kapan saja. Untuk prosedurnya, lihat bagian sebelumnya.

Paket penyebaran setiap revisi dapat mencapai 512 MB (unzip). Karena Anda menyediakan paket deployment di bucket Amazon S3 Anda sendiri dan Lambda mereferensikan kode Anda langsung dari bucket, tidak ada kuota penyimpanan kode terpisah untuk Fungsi Web. Lihat perinciannya di the section called “Kode di Amazon S3”.

Mengkonfigurasi pengaturan revisi

Semua konfigurasi diatur per revisi dan tidak dapat diubah setelah pembuatan. Untuk mengubah pengaturan apa pun, buat revisi baru dengan nilai yang diperbarui. Pengaturan berikut mengontrol perilaku eksekusi.

Untuk pengaturan batas waktu permintaan dan variabel lingkungan, lihat the section called “Batas waktu” dan the section called “Variabel lingkungan” masukthe section called “Mengkonfigurasi Fungsi Web”.

Konkurensi per lingkungan

`maxConcurrencyPerEnvironment`Pengaturan mengontrol berapa banyak permintaan HTTP yang ditangani oleh lingkungan eksekusi tunggal pada saat yang bersamaan. Nilai yang valid adalah 1-128. Defaultnya adalah 64.

Untuk I/O-bound aplikasi (banyak permintaan menunggu database atau panggilan jaringan), nilai yang lebih tinggi memaksimalkan pemanfaatan. Untuk CPU-bound aplikasi, nilai yang lebih rendah mengurangi pertengkaran. Untuk panduan lebih lanjut, lihatthe section called “Penskalaan”.

Titik akhir Fungsi Web Lambda

Endpoint adalah sumber daya turunan dari Fungsi Web yang menyediakan titik masuk jaringan untuk lalu lintas HTTP. Setiap endpoint menyertakan nama domain unik dengan TLS dan jenis otentikasi. Ini juga memiliki pengaturan untuk lingkup geografis, penskalaan, pelambatan, dan perutean lalu lintas. Fungsi Web dapat memiliki hingga 10 titik akhir, masing-masing dengan konfigurasi independen.

Bagian berikut menjelaskan jenis titik akhir, nama domain, otentikasi, dan replikasi. Untuk informasi tentang pengaturan penskalaan dan pembatasan, lihat. the section called “Penskalaan” Untuk informasi tentang mode penyebaran otomatis dan perutean lalu lintas, lihat. the section called “Pola deployment”

Jenis titik akhir

Jenis endpoint menentukan bagaimana Lambda menyebarkan Fungsi Web Anda Wilayah AWS di seluruh dan bagaimana lalu lintas diarahkan ke sana. Anda memilih tipe endpoint saat membuat endpoint. Jenis titik akhir tidak dapat diubah setelah pembuatan.

HomeRegion

> Menjalankan Fungsi Web Anda dalam satu Wilayah AWS. Lambda menyajikan permintaan dari Wilayah tempat Anda membuat titik akhir. Lambda mengalokasikan satu nama domain regional untuk titik akhir.
>
> Gunakan HomeRegionuntuk aplikasi yang tidak memerlukan ketersediaan Multi-region, atau ketika Anda ingin memulai dengan konfigurasi yang paling sederhana.

MultiRegion

Menjalankan Fungsi Web Anda dalam beberapa Wilayah AWS. Anda menentukan Wilayah target saat membuat titik akhir. Lambda menambahkan Wilayah rumah secara otomatis. Titik akhir harus mencakup setidaknya satu Wilayah lainnya. Untuk merutekan lalu lintas ke revisi, Anda menetapkan bobot revisi titik akhir. Lambda kemudian mereplikasi kode dan konfigurasi Anda ke semua Wilayah tertentu secara otomatis. Lambda mengalokasikan satu nama domain global. Ini menggunakan perutean berbasis latensi untuk mengirim setiap permintaan ke Wilayah yang biasanya memberikan latensi terendah untuk pemanggil. Lambda tidak melakukan failover berbasis kesehatan.

Anda mengelola Fungsi Web dari satu Wilayah rumah. Lambda menangani replikasi ke Wilayah lain. Klien terhubung ke satu nama domain.

Gunakan MultiRegionuntuk aplikasi global yang membutuhkan latensi rendah bagi pengguna di beberapa lokasi geografis.

PerRegion

Menjalankan Fungsi Web Anda dalam beberapa Wilayah AWS, dengan nama domain regional terpisah di setiap Wilayah. Anda menentukan Wilayah target saat membuat titik akhir. Lambda menambahkan Wilayah rumah secara otomatis. Titik akhir harus mencakup setidaknya satu Wilayah lainnya. Lambda mengatur titik akhir di setiap Wilayah yang ditentukan secara otomatis. Setiap Wilayah memiliki nama domain independennya sendiri.

Tidak seperti MultiRegion, Lambda tidak melakukan routing berbasis latensi otomatis. Anda mengelola bagaimana lalu lintas mencapai setiap domain regional, misalnya dengan kebijakan perutean Route 53 Anda sendiri atau CloudFront distribusi.

Gunakan PerRegionsaat Anda memerlukan kontrol atas keputusan perutean per Wilayah, misalnya untuk menerapkan batasan geografis, logika failover kustom, atau Region-specific kebijakan lalu lintas.

Jenis Wilayah dan titik akhir tidak dapat diperbarui setelah pembuatan. Untuk mengubahnya, buat titik akhir baru dengan konfigurasi yang Anda inginkan, migrasi lalu lintas ke nama domain titik akhir baru, lalu hapus titik akhir lama.

Untuk detail tentang cara Lambda mereplikasi Fungsi Web Anda di seluruh Wilayah dan rute lalu lintas, lihat. the section called “Multi-Region penyebaran”

Nama domain

Lambda mengalokasikan nama domain unik untuk setiap titik akhir saat Anda membuatnya. Klien mengirim permintaan HTTPS ke domain ini untuk menjalankan Fungsi Web Anda. Domain mendukung HTTPS, dan Lambda mengelola sertifikat TLS untuk domain tersebut. Anda tidak membuat, memperbarui, atau mengelola sertifikat sendiri.

Format nama domain tergantung pada jenis titik akhir:

- HomeRegiondan PerRegion— Setiap Wilayah memiliki domain dalam format:

  ```
  domain-id.shard-id.lambda-web.region.on.aws
  ```

  Contoh: `bcdfghjklmnpqrst.b7t.lambda-web.ap-northeast-3.on.aws`

- MultiRegion— Domain global tunggal dalam format:

  ```
  domain-id.lambda-web.global.on.aws
  ```

  Contoh: `bcdfghjklmn.lambda-web.global.on.aws`

  Lambda menggunakan perutean berbasis latensi untuk memilih Wilayah untuk setiap klien.

Anda dapat mengambil nama domain dari **`GetWebFunctionEndpoint`** respons atau dari konsol Lambda setelah titik akhir mencapai status Aktif.

> **Note**
>
> Endpoint Web Function tidak mendukung melampirkan nama domain kustom secara langsung. Klien yang memanggil domain endpoint harus menggunakan nama Lambda-allocated domain. Untuk melayani lalu lintas di domain Anda sendiri, letakkan CloudFront distribusi Amazon di depan titik akhir dan gunakan nama Lambda-allocated domain sebagai asal.

Jenis autentikasi

`authType`Pengaturan mengontrol apakah Lambda mengautentikasi permintaan HTTP yang masuk sebelum meneruskannya ke aplikasi Anda. Anda dapat memperbarui pengaturan ini pada titik akhir yang ada.

ApplicationManaged

Lambda tidak mengautentikasi permintaan yang masuk. Semua permintaan HTTP diteruskan langsung ke aplikasi Anda. Anda bertanggung jawab untuk menangani otentikasi dalam kode aplikasi Anda.

IamAuth

Lambda memvalidasi tanda tangan setiap permintaan. Menandatangani permintaan dengan AWS Signature Version 4 (SigV4) untuk HomeRegiondan PerRegiondomain, dan dengan Sigv4a Versi Tanda Tangan 4a (Sigv4a) untuk domain global. MultiRegion Hanya penelepon dengan `lambda:InvokeWebFunctionEndpoint` izin yang dapat memanggil titik akhir.

Untuk detail selengkapnya tentang autentikasi, termasuk CloudFront integrasi dan kebijakan berbasis sumber daya, lihat. the section called “Kontrol akses”

Auto-deployment modus

`autoDeploymentMode`Pengaturan mengontrol apakah Lambda secara otomatis merutekan lalu lintas ke revisi baru setiap kali Anda menerapkan aplikasi. Anda dapat memperbarui pengaturan ini pada titik akhir yang ada.

Jika Anda menghilangkan `autoDeploymentMode` permintaan API, defaultnya adalah. `Disabled` Konsol Lambda dan `aws lambda-web deploy` perintah digunakan `LatestRevision` untuk titik akhir baru`HomeRegion`. `LatestRevision`didukung hanya untuk `HomeRegion` titik akhir, jadi `MultiRegion` dan `PerRegion` titik akhir selalu digunakan. `Disabled` Saat menggunakan`Disabled`, tentukan `revisionWeights` total 100.

LatestRevision

Lambda secara otomatis merutekan 100% lalu lintas ke revisi aktif terbaru. Gunakan untuk lingkungan pengembangan dan pementasan. LatestRevision mode hanya tersedia untuk HomeRegiontitik akhir.

Dinonaktifkan

Lambda tidak mengubah perutean lalu lintas saat Anda membuat revisi baru. Anda mengontrol perutean secara manual dengan bobot revisi. Gunakan untuk lingkungan produksi. Mode dinonaktifkan tersedia untuk semua jenis titik akhir. MultiRegiondan PerRegiontitik akhir selalu menggunakan mode Dinonaktifkan.

Untuk detail tentang pola penerapan, rollback, dan perilaku peralihan mode, lihat. the section called “Pola deployment”

Penskalaan dan pelambatan

Setiap titik akhir memiliki pengaturan penskalaan dan pelambatan yang mengontrol kapasitas. Anda dapat memperbarui pengaturan ini pada titik akhir yang ada dengan`update-web-function-endpoint`. Menghilangkan `--scaling-config` atau `--throttle-config` menyimpan nilai saat ini, atau berikan objek kosong (`'{}'`) untuk menghapus nilai yang ditetapkan sebelumnya. Untuk menghapus batas tarif, berikan `--throttle-config` objek kosong. A `rateLimit` dari 0 memblokir semua lalu lintas.

- **`maxEnvironments`** – Opsional. Jumlah maksimum lingkungan eksekusi untuk titik akhir di setiap Wilayah. Ketika tidak disetel, titik akhir menskalakan sesuai kebutuhan dalam kuota akun.
- **`rateLimit`**— Permintaan maksimum per detik. Pengaturan ini bersifat opsional. Jika tidak diatur, titik akhir dibatasi oleh kuota tingkat permintaan per endpoint sebesar 10.000 RPS dan kuota akun. Batas tarif berlaku untuk titik akhir secara keseluruhan, di semua lingkungan eksekusi. Ini bukan batas per-lingkungan. Untuk MultiRegiondan PerRegiontitik akhir, setiap Wilayah memberlakukan batas secara independen. Permintaan berlebih menerima HTTP 429.

Untuk detail tentang cara setelan ini berinteraksi dengan konkurensi tingkat revisi dan panduan dalam memilih nilai, lihat. the section called “Penskalaan”

Membuat dan mengelola titik akhir

Membuat titik akhir

Untuk membuat HomeRegion titik akhir (AWS CLI)

- ```
  aws lambda-web create-web-function-endpoint \
      --function-name my-web-app \
  ```
  ```

```
    --endpoint-name my-endpoint \
    --endpoint-type HomeRegion \
    --auth-type ApplicationManaged \
    --auto-deployment-mode LatestRevision
```

Untuk membuat MultiRegion titik akhir (AWS CLI)

- ```
  aws lambda-web create-web-function-endpoint \
      --function-name my-web-app \
      --endpoint-name my-global-endpoint \
      --endpoint-type MultiRegion \
      --regions us-east-1 eu-west-1 ap-northeast-1 \
      --auth-type ApplicationManaged \
      --revision-weights '[{"revisionId": "rev-a1b2c", "weight": 100}]'
  ```

  MultiRegion dan PerRegion titik akhir memerlukan bobot revisi, karena LatestRevisionpenerapan otomatis tidak tersedia untuk mereka. Untuk informasi selengkapnya, lihat the section called “Pola deployment”.

Mencantumkan titik akhir

```
aws lambda-web list-web-function-endpoints \
    --function-name my-web-app
```

Respons mencakup nama, jenis, nama domain, status, dan konfigurasi setiap titik akhir.

Mendapatkan detail titik akhir

```
aws lambda-web get-web-function-endpoint \
    --function-name my-web-app \
    --endpoint-name my-endpoint
```

Respons tersebut mencakup nama domain titik akhir, status, jenis otentikasi, mode penerapan otomatis, pengaturan penskalaan, bobot revisi, dan (untuk dan `PerRegion` titik akhir) setiap status Wilayah `MultiRegion` dan status pembaruan di bidang. `regionalEndpoints`

Memperbarui titik akhir

Anda dapat memperbarui properti berikut pada titik akhir yang ada:
```

- `description`
- `authType`
- `autoDeploymentMode`
- `revisionWeights`
- `scalingConfig`
- `throttleConfig`

Anda tidak dapat memperbarui properti berikut. Untuk mengubahnya, buat titik akhir baru dengan konfigurasi yang Anda inginkan, migrasi lalu lintas ke nama domain titik akhir baru, lalu hapus titik akhir lama.

- `endpointType` (HomeRegion, MultiRegion, PerRegion)
- `regions`(Wilayah target titik akhir)

Menghapus titik akhir

```
aws lambda-web delete-web-function-endpoint \
    --function-name my-web-app \
    --endpoint-name my-endpoint
```

Ketika Anda menghapus titik akhir, itu memasuki status Menghapus. Lambda berhenti merutekan permintaan baru ke titik akhir dan menghentikan nama domainnya. Untuk `PerRegion` titik akhir, Lambda menghentikan nama domain di setiap Wilayah. Lambda tidak pernah menetapkan nama domain endpoint yang dihapus ke titik akhir lain di akun mana pun. Lambda menguras permintaan dalam penerbangan. Ini memungkinkan permintaan yang sudah dalam proses selesai dan menolak permintaan baru ke nama domain titik akhir. Untuk `MultiRegion` dan `PerRegion` titik akhir, Lambda menghapus titik akhir dari semua Wilayah. Setelah penghapusan selesai, Lambda secara permanen menghapus titik akhir.

Menghapus titik akhir tidak menghapus Fungsi Web atau revisinya. Revisi tetap tersedia untuk digunakan oleh titik akhir lainnya.

Membuat Fungsi Web Lambda

Untuk menerapkan aplikasi dengan Lambda Web Functions, Anda membuat Fungsi Web dengan revisi dan titik akhir untuk menerima lalu lintas. Konsol dan `aws lambda-web deploy` perintah

dapat membuat Fungsi Web, revisi pertamanya, dan titik akhir pertamanya. Revisi adalah snapshot yang tidak dapat diubah dari kode dan konfigurasi Anda. Titik akhir merutekan permintaan ke revisi aktif. Untuk informasi lebih lanjut tentang sumber daya ini, lihat the section called “Konsep inti”. Topik ini menunjukkan cara membuatnya menggunakan konsol Lambda dan. AWS CLI

Prasyarat

Sebelum Anda mulai, pastikan Anda memiliki yang berikut:

- AWS Akun. Untuk membuatnya, lihat Membandingkan opsi pendaftaran di Panduan Referensi Manajemen AWS Akun.
- Izin untuk mengelola Fungsi Web, meneruskan peran eksekusi ke Lambda `iam:PassRole` (), dan untuk membuat atau menggunakan sumber daya Amazon S3 dan IAM yang mendukung.

Konsol membuat atau menggunakan kembali sumber daya yang dibutuhkan Fungsi Web Anda, termasuk peran eksekusi dan bucket Amazon S3 untuk kode Anda.

Untuk mengikuti AWS CLI prosedur, Anda juga perlu:

- AWS CLI versi 2, dalam rilis yang mencakup `aws lambda-web` perintah. Untuk memeriksa, jalankan`aws lambda-web help`. Jika Anda menginstal AWS CLI menggunakan penginstal resmi atau menginstal skrip, jalankan `aws update` untuk memperbarui ke versi terbaru. Jika tidak, lihat Menginstal atau memperbarui ke versi terbaru AWS CLI.
- AWS kredensil dan Wilayah AWS dikonfigurasi untuk file. AWS CLI
- Untuk`aws lambda-web deploy`, izin untuk meneleponsts:GetCallerIdentity, `s3:ListBucket` pada bucket penyebaran, dan, `s3:PutObjects3:GetObject`, dan `s3:GetObjectVersion` pada objeknya.
- Jika `aws lambda-web deploy` membuat sumber daya default, izin untuk memanggil`iam:GetRole`,`iam:CreateRole`,`iam:PutRolePolicy`,`s3:CreateBucket`,`s3:PutBucket` dan`s3:PutBucketVersioning`.
- (Opsional) Node.js 24 atau lebih baru untuk pembangunan lokal. Untuk menginstalNode.js, lihat Unduh Node.js di situs Node.js web.

Membuat Fungsi Web (konsol)

Gunakan konsol Lambda untuk membuat Fungsi Web, revisi pertamanya, dan titik akhir pertamanya untuk menerima lalu lintas.

Untuk membuat Fungsi Web dan titik akhir (konsol)

1. Buka konsol Lambda di. https://console.aws.amazon.com/lambda/
2. Di panel navigasi, pilih Fungsi Web.
3. Pilih Buat Fungsi Web.
4. Di bagian Informasi dasar, lakukan hal berikut:
   a. Untuk nama Fungsi, masukkan nama untuk Fungsi Web Anda.
   b. Untuk Runtime, pertahankan runtime default.
   c. Untuk sumber Kode, simpan template Hello World untuk memulai dari aplikasi contoh yang telah dikonfigurasi sebelumnya. Untuk menerapkan kode Anda sendiri, pilih Unggah file ZIP atau Amazon S3.
5. Di bawah konfigurasi Endpoint, lakukan hal berikut:
   a. Untuk nama Endpoint, masukkan nama untuk titik akhir.
   b. Untuk Tipe, simpan Home-Region. Home-Region Titik akhir berjalan di satu Wilayah, yang menjadikannya konfigurasi paling sederhana untuk Fungsi Web pertama Anda. Untuk opsi Multi-wilayah, lihatthe section called “Titik akhir”.

      > **Note**
      >
      > Anda tidak dapat mengubah jenis titik akhir setelah pembuatan.

   c. Untuk jenis otentikasi, pilih Application-managed otentikasi. Dengan opsi ini, Lambda meneruskan semua permintaan ke aplikasi Anda, sehingga aplikasi Anda menangani autentikasi. Untuk membatasi akses, pilih autentikasi IAM, yang mengharuskan penelepon untuk menandatangani permintaan. Permintaan ke `HomeRegion` dan `PerRegion` titik akhir menggunakan AWS Signature Version 4 (SigV4). Permintaan ke `MultiRegion` titik akhir menggunakan Signature Version 4a (Sigv4a). Untuk informasi selengkapnya, lihat the section called “Kontrol akses”.
   d. Untuk penyebaran Otomatis, simpan revisi terbaru. Dengan mode ini, Lambda secara otomatis merutekan semua lalu lintas ke setiap revisi baru setelah menjadi Aktif. Untuk kontrol lalu lintas manual dengan bobot revisi, pilih Dinonaktifkan. Untuk informasi selengkapnya, lihat the section called “Pola deployment”.
6. (Opsional) Perluas Konfigurasi tambahan untuk mengganti pengaturan default. Secara default, Lambda membuat peran eksekusi dengan izin untuk mengunggah log ke Amazon CloudWatch,

dan bucket Amazon S3 di akun Anda untuk menyimpan kode Anda. Di bagian ini, Anda dapat memilih peran IAM yang ada atau bucket Amazon S3 Anda sendiri, dan Anda juga dapat mengonfigurasi enkripsi dengan kunci yang dikelola pelanggan, konkurensi maksimum per lingkungan, variabel lingkungan, dan tag.

7. Pilih Buat Fungsi Web.

Ketika Anda memilih Create Web Function, Lambda membuat sumber daya Web Function bersama dengan revisi pertamanya, dan menciptakan endpoint pada saat yang sama. Revisi dimulai dalam status Pending sementara Lambda mengunduh kode Anda dan menyiapkan artefak kode darinya, dan kemudian beralih ke Aktif. Titik akhir juga dimulai dalam status Pending. Fungsi Web, revisinya, dan titik akhirnya masing-masing maju melalui negara-negara ini secara independen. Untuk informasi selengkapnya tentang revisi dan status siklus hidup, lihat. the section called “Konsep inti”

Titik akhir mulai melayani lalu lintas ketika mencapai Aktif dan ada revisi Aktif untuk rute ke. Untuk menemukan nama domain endpoint, pada halaman detail fungsi, pilih tab Endpoints, lalu pilih endpoint. Untuk menyajikan aplikasi di domain Anda sendiri, letakkan CloudFront distribusi Amazon di depan titik akhir. Untuk informasi selengkapnya tentang jenis titik akhir dan nama domain, lihatthe section called “Titik akhir”.

Untuk mengonfirmasi bahwa aplikasi Anda menyajikan lalu lintas, kirim permintaan ke nama domain. Karena Anda memilih Application-managed otentikasi, Anda dapat mengirim permintaan biasa tanpa menandatanganinya.

```
curl https://your-endpoint-domain-name/
```

Respons yang berhasil berarti Fungsi Web Anda menerima lalu lintas dan aplikasi Anda menyajikan permintaan.

Membuat Fungsi Web (AWS CLI)

Buat dan terapkan Fungsi Web dari terminal Anda. Perintah starter menghasilkan server Node.js HTTP kecil, menyebarkannya, dan mengembalikan URL titik akhir HTTPS.

Menyebarkan dan menguji Fungsi Web Anda

Untuk menyebarkan dan menguji fungsi Web starter

1. Jalankan perintah berikut:

```
aws lambda-web deploy --name my-web-app --hello-world
```

`--hello-world`menulis file starter ke`./my-web-app`, membuat direktori jika tidak ada. Starter berisi`server.ts`, server TypeScript HTTP, dan`package.json`,`README.md`, dan`AGENTS.md`.

2. Saat penerapan selesai, salin URL titik akhir dari output. URL terlihat mirip dengan berikut ini:

```
https://bcdfghjklmnpqrst.b7t.lambda-web.us-east-1.on.aws
```

3. Kirim permintaan ke URL titik akhir yang Anda salin di langkah sebelumnya. Ganti *`endpoint-url`* dengan URL tersebut:

```
curl endpoint-url
```

Starter mengembalikan respons yang mirip dengan berikut ini:

```
{"message":"Hello from Lambda Web!","path":"/"}
```

Secara default, titik akhir starter menerima permintaan tanpa otentikasi. Untuk meminta otorisasi IAM, tambahkan. `--auth-type IamAuth` Untuk informasi tentang mengontrol akses ke titik akhir Anda, lihatthe section called “Kontrol akses”.

Apa yang dibuat perintah

Pada penyebaran pertama, perintah:

- Membuat atau menggunakan kembali bucket Amazon S3 dengan versi diaktifkan untuk paket penerapan.
- Membuat atau menggunakan kembali peran eksekusi IAM dengan izin Amazon CloudWatch Logs.
- Membuat Fungsi Web dan revisi pertamanya.
- Membuat `HomeRegion` titik akhir bernama `dev` dan menunggu sampai menjadi. `Active` Untuk menggunakan nama yang berbeda, tambahkan`--endpoint-name`.

Fungsi Web berisi revisi kode dan konfigurasi yang tidak dapat diubah. Pada penerapan selanjutnya, perintah membuat revisi lain. `dev`Titik akhir menggunakan `LatestRevision` mode dan melayani revisi setelah menjadi aktif.

Menyebarkan perubahan

Setelah Anda mengedit aplikasi starter, jalankan:

```
aws lambda-web deploy \
  --name my-web-app \
  --code ./my-web-app
```

Menjalankan perintah ini mengunggah paket Anda yang diperbarui, membuat revisi, dan menunggu revisi menjadi. `Active` devTitik akhir telah `autoDeploymentMode` diatur ke `LatestRevision` dan melayani revisi secara otomatis. URL-nya tetap sama.

Untuk menerapkan aplikasi Anda sendiri alih-alih starter, teruskan direktorinya ke`--code`. Server Anda harus mendengarkan pada host dan port dalam variabel `AWS_LAMBDA_HTTP_ENDPOINT` lingkungan. Secara default, runtime dimuat `index.js` di root paket penerapan. Untuk menggunakan file entri yang berbeda, atur variabel `AWS_LAMBDA_NODEJS_ENTRYPOINT` lingkungan. Lambda tidak memverifikasi bahwa server Anda mulai mendengarkan saat membuat revisi. Akibatnya, revisi dan titik akhir dapat ditampilkan `Active` meskipun permintaan gagal. Untuk informasi selengkapnya, lihat the section called “Runtime”.

Gunakan yang sudah ada AWS sumber daya

Jika Anda sudah mengelola bucket Amazon S3 atau peran eksekusi IAM, gunakan perintah berikut untuk menggunakan kembali resource tersebut alih-alih membuat default. Bucket harus mengaktifkan versi dan kebijakan bucket yang memberikan prinsip layanan Lambda () dan. `lambda.amazonaws.com` `s3:GetObject` `s3:GetObjectVersion` Perintah tidak mengubah konfigurasi bucket yang Anda tentukan. Untuk persyaratan bucket, lihatthe section called “Kebijakan bucket untuk Lambda”. Ganti contoh nilai sumber daya dengan milik Anda sendiri.

```
aws lambda-web deploy \
  --name my-web-app \
  --code ./my-web-app \
  --bucket-name amzn-s3-demo-bucket \
  --execution-role-arn arn:aws:iam::111122223333:role/my-web-app-execution-role
```

Peran eksekusi harus memungkinkan Lambda untuk mengambilnya dan harus memberikan izin yang dibutuhkan aplikasi Anda. Untuk informasi selengkapnya, lihat the section called “Peran eksekusi”.

Anda dapat menghilangkan salah satu opsi sumber daya untuk membiarkan `deploy` membuat default. Saat `deploy` membuat Fungsi Web tanpa`--hello-world`, atau menambahkan titik akhir,

ia meminta Anda untuk menekan Enter untuk mengonfirmasi. Dalam skrip atau CI/CD pipeline, tambahkan `--create` untuk mengonfirmasi tanpa prompt.

Gunakan perintah API secara langsung

`deploy`Perintah mengunggah kode Anda ke Amazon S3 dan membuat Fungsi Web, revisi, dan titik akhir untuk Anda. Untuk mengontrol sendiri setiap langkah, gunakan AWS CLI perintah yang memetakan ke operasi API, seperti`create-web-function`,`create-web-function-revision`, dan`update-web-function-endpoint`. Dalam perintah berikut, ganti setiap placeholder input pengguna dengan informasi Anda sendiri.

Untuk membuat Fungsi Web dengan perintah API (AWS CLI)

1. Buat paket deployment .zip dari direktori aplikasi Anda, dengan file entri Anda di root paket. Unggah paket ke bucket Amazon S3 yang memenuhi persyaratan di. the section called “Prasyarat” Perintah berikut membuat paket dan menggunakan AWS CLI untuk mengunggahnya.

   ```
   cd my-app
   zip -r ../my-app.zip .
   aws s3 cp ../my-app.zip s3://amzn-s3-demo-bucket/my-app.zip
   ```

2. Gunakan AWS CLI perintah berikut untuk membuat Web Function dengan revisi pertama dan endpoint. Dalam`revisionConfig`, `buildConfig` menentukan lokasi kode dan runtime, dan `serviceConfig` menentukan peran eksekusi. `endpointConfig`Properti mendefinisikan titik akhir.

   ```
   aws lambda-web create-web-function \
       --function-name my-web-app \
       --revision-config '{
         "buildConfig": {
           "codeConfig": {
             "s3Object": { "bucket": "amzn-s3-demo-bucket", "key": "my-app.zip" }
           },
           "runtimeConfig": { "runtime": "nodejs24.x" }
         },
         "serviceConfig": {
           "executionRoleArn": "arn:aws:iam::111122223333:role/my-web-app-execution-
   role"
         }
       }' \
   ```

```
    --endpoint-config '{
      "endpointName": "my-endpoint",
      "endpointType": "HomeRegion",
      "authType": "ApplicationManaged",
      "autoDeploymentMode": "LatestRevision"
    }'
```

`autoDeploymentMode`Properti ini opsional. Jika Anda menghilangkannya, defaultnya adalah`Disabled`, dan titik akhir tidak merutekan lalu lintas ke revisi baru secara otomatis. Perhatikan revisi dalam tanggapan. `revisionId`

3. Tunggu sampai revisi selesai`Active`. Ganti *`rev-abc12`* dengan ID revisi dari langkah sebelumnya.

```
aws lambda-web wait web-function-revision-active \
    --function-name my-web-app \
    --revision-id rev-abc12
```

4. Dapatkan detail titik akhir.

```
aws lambda-web get-web-function-endpoint \
    --function-name my-web-app \
    --endpoint-name my-endpoint
```

Ketika titik akhir `state` adalah`Active`, kirim permintaan ke nya`domainName`. Responsnya terlihat seperti kutipan berikut.

```
{
    "endpointName": "my-endpoint",
    "endpointType": "HomeRegion",
    "domainName": "bcdfghjklmnpqrst.b7t.lambda-web.us-east-1.on.aws",
    "authType": "ApplicationManaged",
    "autoDeploymentMode": "LatestRevision",
    "state": "Active",
    "updateStatus": "Successful"
}
```

Untuk menerapkan perubahan dengan perintah API (AWS CLI)

- Unggah paket baru dan buat revisi. Revisi adalah snapshot lengkap, jadi lewati penuh `buildConfig` dan `serviceConfig` setiap kali.

```
aws s3 cp ../my-app.zip s3://amzn-s3-demo-bucket/my-app.zip
aws lambda-web create-web-function-revision \
    --function-name my-web-app \
    --build-config '{
      "codeConfig": {
        "s3Object": { "bucket": "amzn-s3-demo-bucket", "key": "my-app.zip" }
      },
      "runtimeConfig": { "runtime": "nodejs24.x" }
    }' \
    --service-config '{
      "executionRoleArn": "arn:aws:iam::111122223333:role/my-web-app-execution-
role"
    }'
```

  Karena titik akhir telah `autoDeploymentMode` ditetapkan`LatestRevision`, ia merutekan semua lalu lintas ke revisi baru setelah revisi menjadi. `Active` Jika Anda mengatur `autoDeploymentMode` ke`Disabled`, Anda merutekan lalu lintas dengan `update-web-function-endpoint` dan merevisi bobot sebagai gantinya. Untuk informasi selengkapnya, lihat the section called “Pola deployment”.

Bersihkan

Saat Anda tidak lagi membutuhkan Fungsi Web, hapus:

```
aws lambda-web delete-web-function --function-name my-web-app
```

Perintah ini juga menghapus endpoint dan revisi Web Function. Ini tidak menghapus sumber daya yang `deploy` dibuat atau digunakan: peran eksekusi IAM (secara `awscli-lambdaweb-my-web-app` default), paket penerapan `lambda-web/my-web-app/` di bucket Amazon S3 `awscli-lambdaweb-`*`account-id`*`-`*`region`*`-an` (secara default), termasuk versi objek yang tidak terkini, atau grup log CloudWatch Amazon Logs untuk Fungsi Web. Hapus sumber daya tersebut secara terpisah hanya jika Anda tidak lagi membutuhkannya. Bucket Amazon S3 yang dikelola dapat dibagikan oleh penerapan lain di wilayah dan wilayah yang sama. Akun AWS

Menerapkan Fungsi Web

Anda menerapkan Fungsi Web dengan memberikan kode aplikasi Anda sebagai paket.zip di bucket Amazon S3. Saat Anda menerapkan revisi baru, Anda mengontrol bagaimana lalu lintas bergerak ke sana dengan pola penerapan seperti kenari dan biru-hijau. Untuk aplikasi global, Anda dapat menerapkan beberapa di Wilayah AWS belakang satu titik akhir, dan Lambda merutekan setiap permintaan ke Wilayah berdasarkan latensi jaringan. Lambda tidak menjamin bahwa setiap permintaan masuk ke Wilayah latensi terendah.

Topik

- Menyebarkan kode Fungsi Web dari Amazon S3
- Pola deployment
- Multi-Region penyebaran

Menyebarkan kode Fungsi Web dari Amazon S3

Anda memberikan kode Fungsi Web Lambda sebagai paket penyebaran.zip di bucket Amazon S3. Ini adalah satu-satunya cara untuk menyediakan kode untuk Fungsi Web. Saat Anda membuat Fungsi Web atau revisi, tentukan objek di `s3Object` bidang, yang bersarang `codeConfig` di bawah permintaan. `buildConfig` Dis`3Object`, sertakan bucket, key, dan versionID opsional. Lambda mereferensikan kode Anda langsung dari ember Anda. Karena Lambda menyematkan setiap revisi ke versi objek tertentu, penerapan Anda dapat direproduksi. Anda dapat menggunakan kembali satu paket penerapan di seluruh akun dan AWS Wilayah.

Topik

- Prasyarat
- Izin yang diperlukan
- Membuat Fungsi Web dengan kode dari Amazon S3
- Siklus hidup objek
- Multi-Region penyebaran

Prasyarat

Sebelum menerapkan kode Fungsi Web dari Amazon S3, pastikan paket bucket dan deployment memenuhi persyaratan berikut:

Versi S3

Bucket harus mengaktifkan versi S3. Lambda menggunakan versi objek untuk melacak versi kode Anda yang digunakan revisi.

ID Versi

`versionId`Parameternya opsional dalam permintaan. Jika Anda menghilangkannya, Lambda menggunakan versi terbaru dari objek pada waktu pembuatan. Either way, Lambda menyematkan revisi ke versi objek tertentu.

Ukuran paket penyebaran maksimum

512 MB (dibuka ritsleting).

Berkas entri

Paket deployment harus berisi file entri aplikasi Anda. Secara default, runtime dimuat `index.js` di root paket. Untuk menggunakan file entri yang berbeda, atur variabel `AWS_LAMBDA_NODEJS_ENTRYPOINT` lingkungan pada revisi ke jalur file relatif terhadap root paket. Setelah memuat file entri, runtime menunggu server HTTP Anda mulai mendengarkan di host dan port dari variabel `AWS_LAMBDA_HTTP_ENDPOINT` lingkungan. Jika runtime tidak dapat menyelesaikan file entri, atau server tidak pernah mulai mendengarkan, inisialisasi gagal dan permintaan ke titik akhir gagal.

Lambda tidak memverifikasi bahwa server Anda mulai mendengarkan saat membuat atau memperbarui titik akhir untuk menggunakan revisi. Revisi dan titik akhir masih dapat ditampilkan `Active` dalam kasus ini. Setelah setiap penerapan, kirim permintaan pengujian dan periksa log aplikasi Anda.

Lokasi ember

Bucket dapat berada di AWS akun yang berbeda dan di AWS Wilayah yang berbeda dari Fungsi Web. Bucket harus berada di AWS partisi yang sama dengan Fungsi Web. Bucket tidak dapat berada di Wilayah keikutsertaan (Wilayah yang tidak diaktifkan secara default). Jika salah satu kondisi tidak terpenuhi, Lambda menolak permintaan tersebut. Cross-Region pengambilan objek menimbulkan biaya transfer data Amazon S3 tambahan. Untuk informasi selengkapnya tentang harga, lihat Harga Amazon S3.

Izin yang diperlukan

Untuk menerapkan kode Fungsi Web dari Amazon S3, Anda harus mengonfigurasi izin untuk identitas panggilan dan prinsip layanan Lambda.

Izin penelepon

Untuk membuat Fungsi Web atau revisi, identitas IAM Anda memerlukan izin berikut pada objek S3:

- `s3:GetObject`
- `s3:GetObjectVersion`

Untuk bucket lintas akun, kebijakan bucket juga harus memberikan akses identitas panggilan ke objek.

Kebijakan bucket untuk Lambda

Kebijakan bucket harus memberikan Lambda service principal (`lambda.amazonaws.com`) `s3:GetObject` dan `s3:GetObjectVersion` izin. Lingkup kebijakan dengan `aws:SourceArn` kondisi yang sesuai dengan pola ARN revisi. Gunakan `ArnLike` dengan wildcard untuk ID revisi, karena Lambda menghasilkan ID revisi saat Anda membuat revisi.

Jika objek dienkripsi dengan AWS KMS key (SSE-KMS), kebijakan kunci juga harus mengizinkan `lambda.amazonaws.com` untuk memanggil `kms:Decrypt` kunci itu. Lambda memeriksa akses kepala layanan saat Anda membuat revisi dan menolak permintaan jika akses tidak ada.

Contoh kebijakan bucket berikut memberi Lambda akses untuk membaca paket penerapan Anda untuk setiap revisi Fungsi Web yang ditentukan.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowLambdaWebFunctionCodeAccess",
            "Effect": "Allow",
            "Principal": {
                "Service": "lambda.amazonaws.com"
            },
            "Action": [
                "s3:GetObject",
```

```
                "s3:GetObjectVersion"
            ],
            "Resource": "arn:aws:s3:::my-bucket/*",
            "Condition": {
                "ArnLike": {
                    "aws:SourceArn": "arn:aws:lambda:us-east-1:111122223333:web-
function/my-function/revision/*"
                }
            }
        }
    ]
}
```

Untuk membatasi akses ke key prefix yang menyimpan paket deployment Anda, ubah `Resource` nilainya menjadi, di mana nama bucket Anda`arn:aws:s3:::my-bucket/my-prefix/*`, dan *my-bucket my-prefix* merupakan key prefix. Simpan `/*` di akhir awalan. Jangan ubah `revision/*` wildcard di`aws:SourceArn`.

Membuat Fungsi Web dengan kode dari Amazon S3

Anda menentukan paket penyebaran S3 Anda di `codeConfig` bagian. `buildConfig` Prosedur berikut menunjukkan cara membuat Fungsi Web dan cara membuat revisi baru menggunakan. AWS CLI

Untuk membuat Fungsi Web dengan kode dari Amazon S3 (AWS CLI)

- Jalankan perintah berikut untuk membuat Fungsi Web yang mereferensikan paket deployment .zip Anda di Amazon S3.

```
aws lambda-web create-web-function \
    --function-name my-web-app \
    --revision-config '{
      "buildConfig": {
        "codeConfig": {
          "s3Object": { "bucket": "my-bucket", "key": "my-app.zip" }
        },
        "runtimeConfig": { "runtime": "nodejs24.x" }
      },
      "serviceConfig": {
        "executionRoleArn": "arn:aws:iam::111122223333:role/my-web-function-role"
      }
    }' \
```

```
    --endpoint-config '{
      "endpointName": "my-endpoint",
      "endpointType": "HomeRegion",
      "authType": "ApplicationManaged"
    }'
```

Ketika Anda menghilangkan`versionId`, Lambda menggunakan versi terbaru dari objek pada waktu pembuatan.

Untuk membuat revisi baru dengan versi objek tertentu (AWS CLI)

- Jalankan perintah berikut untuk membuat revisi baru yang menggunakan versi tertentu dari paket penyebaran Anda.

```
aws lambda-web create-web-function-revision \
    --function-name my-web-app \
    --build-config '{
      "codeConfig": {
        "s3Object": {
          "bucket": "my-bucket",
          "key": "my-app.zip",
          "versionId": "abc123def456"
        }
      },
      "runtimeConfig": { "runtime": "nodejs24.x" }
    }' \
    --service-config '{
      "executionRoleArn": "arn:aws:iam::111122223333:role/my-web-function-role"
    }'
```

Dengan menentukan`versionId`, Anda menyematkan revisi ke versi objek yang tepat daripada versi terbaru.

Siklus hidup objek

⚠ Jangan hapus objek S3 atau hapus akses Lambda untuk revisi aktif

Lambda mereferensikan kode Anda dari bucket Amazon S3 Anda. Jika Anda menghapus objek S3 yang digunakan revisi, atau versi objek tertentu yang digunakan revisi, revisi akan berpindah ke status `Failed` saat Lambda selanjutnya memverifikasi akses ke objek, dan

kemudian tidak dapat lagi melayani lalu lintas. Simpan versi objek untuk semua revisi yang masih Anda gunakan.
Menghapus akses kepala layanan Lambda ke bucket memiliki efek yang sama. Jika Anda mengubah kebijakan bucket untuk menolak atau menghapus `s3:GetObject` dan `s3:GetObjectVersion` izin untuk`lambda.amazonaws.com`, revisi yang ada yang mereferensikan objek dalam bucket tersebut akan dipindahkan ke status. `Failed` Pertahankan objek dan akses baca kepala layanan Lambda selama revisi apa pun mereferensikan objek tersebut.
`Failed`Negara adalah permanen. Memulihkan objek yang dihapus atau izin bucket tidak mengembalikan revisi ke status. `Active` Untuk memulihkan, buat revisi baru yang mereferensikan versi objek yang tersedia, lalu perbarui bobot revisi titik akhir Anda untuk mengarahkan lalu lintas ke sana. Untuk pola penerapan, lihatthe section called “Pola deployment”.

Lambda tidak secara otomatis mengalihkan lalu lintas dari revisi. `Failed` Permintaan bahwa rute titik akhir ke `Failed` revisi ditolak dengan kode status HTTP 409 dan header `lambda-web-error-type` respons dari. `Platform.RevisionNotActive` Jika titik akhir membebani lalu lintas di beberapa revisi, hanya bagian permintaan yang diarahkan ke revisi yang terpengaruh; permintaan yang diarahkan ke `Failed` revisi sehat disajikan secara normal. Untuk menghentikan kegagalan, perbarui bobot revisi titik akhir untuk menghapus revisi. `Failed`

Menimpa objek dengan unggahan baru membuat versi S3 baru. Ini tidak memengaruhi revisi yang ada: saat Anda membuat revisi, Lambda menyelesaikan dan menyematkan versi objek yang tepat, bahkan ketika Anda menghilangkannya. `versionId` Namun, jika Anda menghapus versi objek lama yang direferensikan revisi, revisi tersebut `Failed` pindah ke status. Mengunggah objek pengganti tidak membantu, karena penggantian adalah versi baru dengan ID versi yang berbeda.

Multi-Region penyebaran

Untuk Fungsi Web dengan `MultiRegion` atau `PerRegion` titik akhir, Lambda membaca objek sumber Anda dari bucket yang Anda tentukan, di Wilayah mana pun bucket berada. Lambda kemudian mereplikasi artefak kode yang disiapkan yang diturunkan dari objek itu ke Wilayah lain. Anda tidak perlu menyalin paket penerapan Anda ke bucket di Wilayah lain. Untuk informasi lebih lanjut tentang apa yang direplikasi Lambda, lihat. the section called “Apa yang direplikasi Lambda”

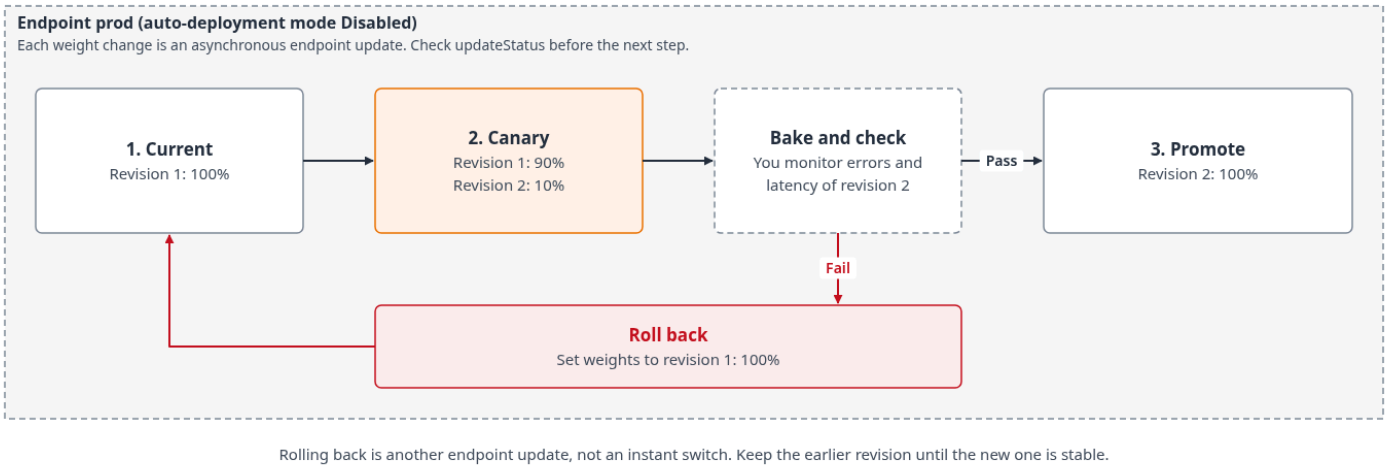
Pola deployment

Untuk merilis versi baru aplikasi Fungsi AWS Lambda Web Anda, Anda membuat revisi baru. Titik akhir dengan `autoDeploymentMode` set untuk LatestRevisionmerutekan semua lalu lintas ke sana setelah menjadi Aktif. Titik akhir dengan `autoDeploymentMode` disetel ke Nonaktif tetap menyajikan revisi mereka saat ini hingga Anda memperbarui bobot revisi. Anda mengontrol bagaimana lalu lintas bergerak dari revisi saat ini ke yang baru menggunakan mode penyebaran otomatis titik akhir dan bobot revisi. Bagian berikut menjelaskan mode penerapan otomatis, pola penerapan umum (kenari, biru-hijau, dan rollback), dan aturan yang mengatur bobot revisi.

Perubahan pada `autoDeploymentMode` dan `revisionWeights` merupakan pembaruan titik akhir asinkron. Tunggu `updateStatus` hingga menjadi `Successful` sebelum Anda mengevaluasi penerapan atau mengirimkan pembaruan titik akhir lainnya. Jika pembaruan gagal, `updateStatusReason` jelaskan penyebabnya.

Diagram berikut menunjukkan penerapan kenari pada titik akhir yang mode penerapan otomatisnya Dinonaktifkan. Setiap perubahan bobot adalah pembaruan endpoint asinkron.

1. Revisi 1 dimulai dengan 100 persen lalu lintas.
2. Anda mengarahkan 10 persen lalu lintas ke revisi 2 dan memantaunya.
3. Jika revisi 2 melewati cek Anda, Anda mengarahkan semua lalu lintas ke sana.
4. Jika revisi 2 gagal, Anda mengatur bobot kembali ke revisi 1 pada 100 persen.



Rolling back is another endpoint update, not an instant switch. Keep the earlier revision until the new one is stable.

Auto-deployment mode

`autoDeploymentMode`Pengaturan pada titik akhir mengontrol apakah Lambda secara otomatis merutekan lalu lintas ke revisi baru saat Anda menerapkan. Anda mengatur ini ketika Anda membuat titik akhir, dan Anda dapat mengubahnya pada titik akhir yang ada.

Jika Anda menghilangkan`autoDeploymentMode`, defaultnya. `Disabled` `LatestRevision`didukung hanya untuk `HomeRegion` titik akhir, jadi `MultiRegion` dan `PerRegion` titik akhir selalu digunakan. `Disabled` Saat menggunakan`Disabled`, Anda mengatur `revisionWeights` titik akhir, seperti yang dijelaskan dalamthe section called “Bobot revisi”.

LatestRevision

Setiap kali revisi baru menjadi Aktif, Lambda merutekan 100% lalu lintas titik akhir ke sana. Anda tidak perlu memperbarui bobot revisi, dan Anda tidak dapat mengatur bobot revisi saat titik akhir dalam mode ini. Gunakan mode ini untuk lingkungan pengembangan dan pementasan di mana Anda menginginkan penerapan otomatis tanpa intervensi manual.

Dinonaktifkan

Lambda tidak mengubah perutean lalu lintas saat Anda membuat revisi baru. Anda mengontrol perutean lalu lintas secara manual dengan mengatur bobot revisi. Gunakan mode ini untuk lingkungan produksi tempat Anda ingin mengontrol peluncuran revisi baru dengan pola penerapan kenari atau biru-hijau.

Untuk mengubah mode penyebaran otomatis (AWS CLI)

1. Gunakan AWS Command Line Interface (AWS CLI) untuk memperbarui titik akhir dan menonaktifkan penerapan otomatis.

```
aws lambda-web update-web-function-endpoint \
    --function-name my-web-app \
    --endpoint-name my-endpoint \
    --auto-deployment-mode Disabled
```

2. Tunggu hingga pembaruan titik akhir selesai.

```
aws lambda-web wait web-function-endpoint-updated \
    --function-name my-web-app \
    --endpoint-name my-endpoint
```

Beralih antar mode

Saat Anda beralih dari LatestRevisionke Dinonaktifkan tanpa menentukan bobot revisi, perutean lalu lintas saat ini tetap tidak berubah setelah pembaruan berhasil. Revisi yang mengarahkan lalu lintas ke titik akhir sebelum pembaruan terus menerima 100% lalu lintas, bahkan setelah Anda membuat revisi yang lebih baru. Itu membuat lalu lintas itu sampai Anda memperbarui bobot revisi. Anda juga dapat mengganti mode dan mengatur bobot revisi dalam permintaan yang sama`UpdateWebFunctionEndpoint`.

Saat Anda beralih dari Dinonaktifkan ke LatestRevision, Lambda memperbarui titik akhir untuk merutekan 100% lalu lintas ke revisi terbaru. Revisi terbaru harus Aktif. Jika Tertunda atau Gagal, permintaan gagal dengan kesalahan konflik. Setelah pembaruan berhasil, bobot revisi yang dikonfigurasi sebelumnya tidak lagi berlaku.

Bobot revisi

Saat mode penyebaran otomatis Dinonaktifkan, Anda mengontrol perutean lalu lintas dengan menetapkan bobot ke revisi. Lambda mendistribusikan lalu lintas antar revisi secara proporsional dengan bobotnya.

Bobot revisi adalah pengaturan pada titik akhir (`revisionWeights`properti), yang Anda ubah. `UpdateWebFunctionEndpoint` Mereka bukan pengaturan pada revisi.

Bobot revisi mengikuti aturan ini:

- Setiap berat adalah bilangan bulat dari 1 hingga 100.
- Bobot dalam permintaan harus berjumlah tepat 100.
- Setiap revisi hanya dapat muncul sekali dalam bobot.
- Setiap revisi dalam bobot harus dalam keadaan Aktif saat Anda mengatur bobot. Jika tidak, API menolak permintaan tersebut.
- Titik akhir merutekan lalu lintas ke maksimum dua revisi pada saat yang bersamaan.
- Tidak ada bobot yang memiliki nilai default. Untuk merutekan semua lalu lintas ke satu revisi, tentukan hanya revisi itu dengan berat 100. Revisi yang tidak dalam bobot tidak menerima lalu lintas dari titik akhir itu.

Pengidentifikasi revisi dihasilkan oleh Lambda saat Anda membuat revisi. Untuk menemukan pengidentifikasi revisi untuk Fungsi Web Anda, gunakan perintah berikut:

```
aws lambda-web list-web-function-revisions \
    --function-name my-web-app
```

Respons mencantumkan revisi secara berurutan dari yang terbaru ke yang terlama, termasuk pengenal, stempel waktu pembuatan, dan status saat ini.

Pola deployment

Pola penerapan berikut menggunakan bobot revisi untuk mengontrol bagaimana lalu lintas bergerak dari revisi Anda saat ini ke revisi baru. Pola ini memerlukan mode penyebaran otomatis titik akhir (pengaturan **`autoDeploymentMode`** titik akhir) untuk Dinonaktifkan. Titik akhir yang dibuat `aws lambda-web deploy` perintah menggunakan LatestRevision. Untuk menggunakan pola ini pada titik akhir itu, pertama-tama alihkan ke Dinonaktifkan, seperti yang ditunjukkan padathe section called "Auto-deployment mode".

Penyebaran kenari

Penyebaran kenari mengirimkan sebagian kecil lalu lintas ke revisi baru sehingga Anda dapat memantau kesalahan sebelum merutekan semua lalu lintas. Jika revisi baru memiliki masalah, hanya sebagian kecil permintaan yang terpengaruh.

Untuk melakukan penyebaran kenari (AWS CLI)

1. Buat revisi baru dengan kode yang diperbarui.

   ```
   aws lambda-web create-web-function-revision \
       --function-name my-web-app \
       --build-config '{
         "codeConfig": {
           "s3Object": { "bucket": "my-bucket", "key": "my-app-v2.zip" }
         },
         "runtimeConfig": { "runtime": "nodejs24.x" }
       }' \
       --service-config '{
         "executionRoleArn": "arn:aws:iam::111122223333:role/my-web-function-role"
       }'
   ```

   Perhatikan pengidentifikasi revisi dalam respons.

   Revisi baru tidak menyalin pengaturan dari revisi saat ini. `serviceConfig`Pengaturan apa pun yang Anda hilangkan mengambil nilai defaultnya, misalnya a `timeoutSeconds` dari 30

dan tidak ada variabel lingkungan. Sertakan setiap pengaturan yang digunakan revisi saat ini, sepertienvironmentVariables,, timeoutSecondsmaxConcurrencyPerEnvironment, dantelemetryConfig. Untuk melihat konfigurasi revisi saat ini, jalankanaws lambda-web get-web-function-revision.

2. Tunggu sampai revisi baru Aktif. API menolak bobot revisi yang mereferensikan revisi yang tidak Aktif.

```
aws lambda-web wait web-function-revision-active \
    --function-name my-web-app \
    --revision-id rev-def45
```

3. Rutekan 10% lalu lintas ke revisi baru dan 90% ke revisi saat ini.

```
aws lambda-web update-web-function-endpoint \
    --function-name my-web-app \
    --endpoint-name my-endpoint \
    --revision-weights '[{"revisionId": "rev-abc12", "weight": 90}, {"revisionId":
 "rev-def45", "weight": 10}]'
```

4. Tunggu hingga pembaruan titik akhir selesai.

```
aws lambda-web wait web-function-endpoint-updated \
    --function-name my-web-app \
    --endpoint-name my-endpoint
```

5. Pantau revisi baru melalui log aplikasi atau metrik khusus. Sertakan pengenal build atau revisi yang Anda kontrol di dalamnya. Metrik Fungsi Web tidak menyertakan dimensi revisi. Akibatnya, mereka tidak dapat memisahkan kesalahan kenari dari kesalahan pada revisi saat ini. Lihat perinciannya di the section called “Metrik-metrik”. Jika kenari memenuhi ambang kesalahan dan latensi yang Anda tentukan sebelum peluncuran, tingkatkan bobotnya.

```
aws lambda-web update-web-function-endpoint \
    --function-name my-web-app \
    --endpoint-name my-endpoint \
    --revision-weights '[{"revisionId": "rev-abc12", "weight": 50}, {"revisionId":
 "rev-def45", "weight": 50}]'
```

6. Tunggu hingga pembaruan titik akhir selesai.

```
aws lambda-web wait web-function-endpoint-updated \
    --function-name my-web-app \
```

```
    --endpoint-name my-endpoint
```

7. Ketika Anda yakin revisi baru itu sehat, arahkan semua lalu lintas ke sana.

```
aws lambda-web update-web-function-endpoint \
    --function-name my-web-app \
    --endpoint-name my-endpoint \
    --revision-weights '[{"revisionId": "rev-def45", "weight": 100}]'
```

8. Tunggu hingga pembaruan titik akhir selesai.

```
aws lambda-web wait web-function-endpoint-updated \
    --function-name my-web-app \
    --endpoint-name my-endpoint
```

Blue-green penyebaran

Penerapan biru-hijau membuat revisi saat ini melayani lalu lintas saat Anda memvalidasi revisi baru, dan kemudian mengalihkan semua lalu lintas ke revisi baru dalam satu pembaruan titik akhir. Prosedur berikut memvalidasi dengan pembagian lalu lintas genap. Untuk memvalidasi tanpa lalu lintas produksi, gunakan titik akhir pengujian terpisah, seperti yang dijelaskan dalam. the section called “Pertimbangan deployment”

Untuk melakukan penyebaran biru-hijau (AWS CLI)

1. Buat revisi baru dengan kode yang diperbarui, dan tunggu hingga aktif. Gunakan perintah yang sama yang ditunjukkan dalam prosedur penyebaran kenari sebelumnya.
2. Pisahkan lalu lintas secara merata di antara dua revisi.

```
aws lambda-web update-web-function-endpoint \
    --function-name my-web-app \
    --endpoint-name my-endpoint \
    --revision-weights '[{"revisionId": "rev-abc12", "weight": 50}, {"revisionId":
 "rev-def45", "weight": 50}]'
```

3. Tunggu hingga pembaruan titik akhir selesai.

```
aws lambda-web wait web-function-endpoint-updated \
    --function-name my-web-app \
    --endpoint-name my-endpoint
```

4. Pantau kedua revisi melalui log aplikasi atau metrik khusus. Sertakan pengenal build atau revisi yang Anda kontrol di dalamnya. Ketika revisi baru memenuhi kriteria kesehatan yang Anda tentukan sebelum peluncuran, arahkan semua lalu lintas ke sana.

```
aws lambda-web update-web-function-endpoint \
    --function-name my-web-app \
    --endpoint-name my-endpoint \
    --revision-weights '[{"revisionId": "rev-def45", "weight": 100}]'
```

5. Tunggu hingga pembaruan titik akhir selesai.

```
aws lambda-web wait web-function-endpoint-updated \
    --function-name my-web-app \
    --endpoint-name my-endpoint
```

Berguling kembali

Jika Anda mendeteksi masalah dengan revisi baru (tingkat kesalahan tinggi, peningkatan latensi), Anda dapat memutar kembali dengan satu pembaruan titik akhir yang merutekan semua lalu lintas ke revisi yang diketahui baik sebelumnya.

> **Bergulir kembali dalam LatestRevision mode**
>
> Meskipun mode penerapan otomatis titik akhir adalah LatestRevision, permintaan yang hanya menetapkan bobot revisi gagal. Prosedur berikut menyetel mode penerapan otomatis ke Dinonaktifkan dan bobot revisi dalam permintaan yang sama, sehingga berfungsi di salah satu mode.
>
> Titik akhir kemudian tetap dalam mode Dinonaktifkan sampai Anda mengubahnya.

Untuk memutar kembali ke revisi sebelumnya (AWS CLI)

1. Perbarui titik akhir untuk merutekan 100% lalu lintas ke revisi yang diketahui baik dan atur mode penerapan otomatis ke Dinonaktifkan.

```
aws lambda-web update-web-function-endpoint \
    --function-name my-web-app \
    --endpoint-name my-endpoint \
    --auto-deployment-mode Disabled \
```

```
    --revision-weights '[{"revisionId": "rev-abc12", "weight": 100}]'
```

2. Tunggu hingga pembaruan titik akhir selesai.

```
aws lambda-web wait web-function-endpoint-updated \
    --function-name my-web-app \
    --endpoint-name my-endpoint
```

Setelah pembaruan berhasil, Lambda berhenti merutekan permintaan baru ke revisi yang bermasalah. In-flight permintaan pada revisi itu selesai secara normal. Untuk informasi selengkapnya tentang status pembaruan titik akhir, lihatthe section called “Status titik akhir”.

Jika revisi yang Anda putar kembali memiliki sedikit atau tidak ada lingkungan eksekusi yang berjalan, Lambda meningkatkannya saat lalu lintas bergeser ke sana. Permintaan dapat menunggu saat lingkungan eksekusi diinisialisasi, yang meningkatkan latensi. Permintaan masih dapat gagal karena kegagalan startup, batas tingkat titik akhir atau lingkungan maksimum, atau kuota vCPU akun. Beberapa kegagalan ini mengembalikan respons HTTP 429.

Pertimbangan deployment

Apa yang memicu revisi baru?

Lambda membuat revisi baru saat Anda menelepon. `CreateWebFunctionRevision` Ini termasuk menerapkan kode aplikasi baru dan mengubah pengaturan konfigurasi seperti batas waktu, konkurensi, atau variabel lingkungan. Setiap perubahan menghasilkan revisi yang tidak dapat diubah.

Dapatkah saya menguji revisi baru sebelum merutekan lalu lintas produksi ke sana?

Ya. Anda hanya dapat menjalankan revisi melalui titik akhir, jadi ada dua pendekatan:

- Titik akhir pengujian khusus - Buat titik akhir terpisah untuk pengujian. Rutekan lalu lintas ke revisi baru pada titik akhir pengujian terlebih dahulu (dengan menyetel bobot revisi titik akhir pengujian), memvalidasinya, dan kemudian memperbarui bobot titik akhir produksi untuk menyertakan revisi baru.
- Bobot minimal — Tetapkan revisi baru bobot 1% pada titik akhir produksi. Ini mengirimkan sebagian kecil lalu lintas produksi ke revisi baru sehingga Anda dapat memantau kesalahan dengan dampak pelanggan minimal.

Apa yang terjadi pada revisi yang tidak ada dalam bobot revisi?

Revisi yang tidak berada dalam bobot revisi titik akhir tidak menerima lalu lintas dari titik akhir tersebut. Titik akhir lain dari Fungsi Web yang sama masih dapat merutekan ke sana. Revisi tetap Aktif sampai Anda menghapusnya atau sampai pembersihan otomatis menghapusnya. Ini pindah ke Gagal jika Lambda kehilangan akses ke objek Amazon S3-nya. Untuk pembersihan otomatis, lihatthe section called “Batas revisi dan pembersihan otomatis”.

Bagaimana cara mengidentifikasi revisi mana yang terbaru?

Panggil `ListWebFunctionRevisions`. Respons mencantumkan revisi dalam urutan menurun`createdAt`, jadi revisi pertama dalam daftar adalah yang terbaru.

Bisakah saya menghapus titik akhir saat merutekan lalu lintas?

Ya. Saat Anda menghapus titik akhir, Lambda berhenti merutekan permintaan baru ke titik akhir tersebut, dan permintaan ke nama domain titik akhir ditolak setelah penghapusan diterapkan. Revisi itu sendiri tidak terpengaruh dan masih dapat digunakan oleh titik akhir lainnya.

Dapatkah saya menghapus revisi saat menerima lalu lintas?

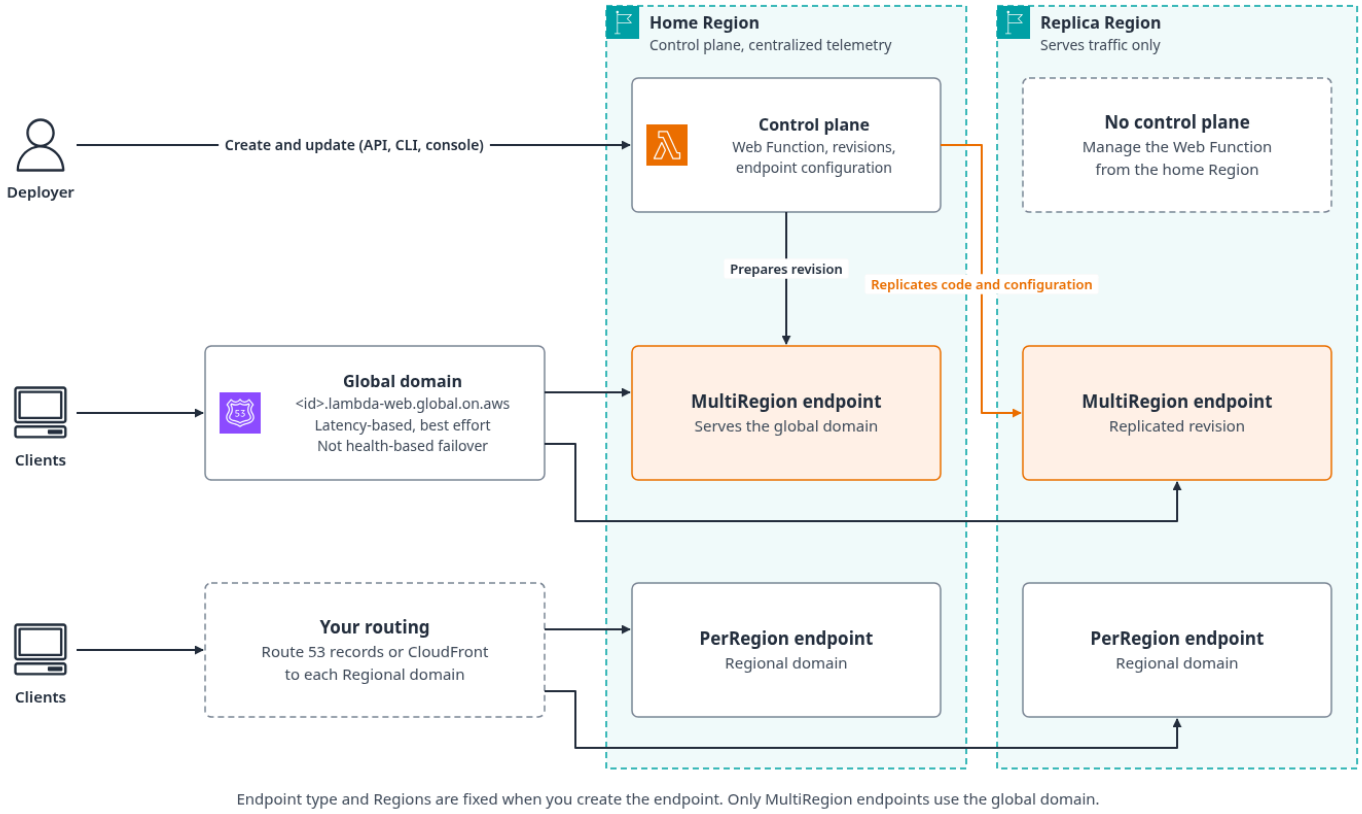
Tidak. Jika ada titik akhir yang merutekan lalu lintas ke revisi, termasuk ketika `regionalEndpoints` properti `MultiRegion` atau `PerRegion` titik akhir mereferensikan revisi, `DeleteWebFunctionRevision` permintaan gagal dengan kesalahan konflik. Untuk menghapus revisi, pertama-tama perbarui bobot revisi titik akhir untuk menghapusnya dari perutean lalu lintas di setiap titik akhir, lalu hapus. Karena Lambda memblokir penghapusan sementara revisi memiliki bobot yang ditetapkan, menghapus revisi tidak pernah mengubah distribusi lalu lintas titik akhir.

Multi-Region penyebaran

Untuk MultiRegiondan PerRegiontitik akhir, Lambda mereplikasi artefak runtime ke semua Wilayah target sehingga aplikasi Anda dapat melayani lalu lintas dekat dengan pengguna akhir. Anda mengelola Fungsi Web sepenuhnya dari Wilayah rumah. Sumber daya yang direplikasi tidak terlihat atau dapat dimodifikasi di Wilayah replika.

Diagram berikut menunjukkan bagaimana bidang kontrol di Wilayah asal menyiapkan revisi dan mereplikasi kode dan konfigurasinya ke Region replika, dan bagaimana klien mencapai dan titik akhir. MultiRegionPerRegion Replika Region tidak memiliki bidang kontrol. MultiRegionTitik akhir memiliki satu domain global yang merutekan berdasarkan latensi, bukan berdasarkan kesehatan Wilayah. Untuk PerRegiontitik akhir, Anda mengelola cara klien mencapai domain Regional di setiap

Wilayah, misalnya dengan catatan Route 53 atau CloudFront. Anda menetapkan tipe titik akhir dan Wilayah saat membuat titik akhir, dan Anda tidak dapat mengubahnya nanti.



Topik

- Apa yang direplikasi Lambda
- Visibilitas replika
- Ketika replikasi terjadi
- Cara kerja routing
- Kegagalan replikasi
- Konsistensi akhirnya

Apa yang direplikasi Lambda

Lambda mereplikasi hal berikut ke setiap Wilayah target:

- Artefak kode yang disiapkan - Artefak runtime internal yang Lambda berasal dari objek sumber Amazon S3 berversi.
- Konfigurasi revisi - Timeout, konkurensi, variabel lingkungan, dan pengaturan runtime.

- Resource-based kebijakan — Kebijakan yang dilampirkan pada Fungsi Web yang memberikan akses ke sana.
- Tag — Tag pada Fungsi Web.

Kuota untuk Fungsi Web berlaku per. Wilayah AWS Setiap Wilayah target memberlakukan kuota akunnya sendiri pada titik akhir yang direplikasi. Misalnya, penskalaan lingkungan eksekusi di Region replika dibatasi oleh kuota vCPU akun Wilayah tersebut. Untuk kuota akun, lihatthe section called "Kuota akun".

Visibilitas replika

Sumber daya yang direplikasi tidak terlihat di Wilayah replika. Bidang kontrol untuk Fungsi Web hanya disajikan di Wilayah asalnya, jadi semua permintaan API harus dikirim ke Wilayah asal. Hal ini mencakup:

- Daftar, dapatkan, dan jelaskan permintaan untuk Fungsi Web, revisinya, dan titik akhirnya.
- Buat, perbarui, dan hapus permintaan untuk fungsi, revisi, dan konfigurasi titik akhir.

Permintaan API yang dikirim ke titik akhir bidang kontrol Region replika gagal, di mana pun penelepon berada. Artefak runtime yang direplikasi hanya ada untuk pesawat data untuk melayani lalu lintas.

Ketika replikasi terjadi

Replikasi dipicu saat Anda membuat atau memperbarui titik PerRegionakhir MultiRegionatau. Secara khusus, replikasi terjadi ketika:

- Anda membuat PerRegion titik akhir MultiRegion atau dengan bobot revisi yang menunjuk ke revisi.
- Anda memperbarui bobot revisi pada PerRegion titik akhir MultiRegion atau untuk menyertakan revisi baru.
- Anda memperbarui pengaturan titik akhir lainnya, sepertiauthType,scalingConfig, atauthrottleConfig, pada titik PerRegion akhir MultiRegion atau.

Membuat revisi saja tidak memicu replikasi. Revisi harus disertakan dalam bobot revisi titik akhir yang memiliki Wilayah target yang dikonfigurasi. Karena hanya mode penyebaran otomatis Dinonaktifkan

yang tersedia untuk MultiRegion dan PerRegion titik akhir, replikasi selalu merupakan hasil dari perubahan titik akhir yang Anda buat.

Cara kerja routing

Perilaku routing tergantung pada tipe endpoint:

MultiRegion

Lambda menyediakan satu nama domain global. Permintaan dirutekan menggunakan routing berbasis latensi. Permintaan dialihkan ke Wilayah yang biasanya menyediakan latensi jaringan terendah untuk penelepon, berdasarkan pengukuran latensi yang dikumpulkan Route 53 dari waktu ke waktu. Karena latensi internet berubah, Lambda tidak menjamin bahwa setiap permintaan masuk ke Wilayah latensi terendah. Lambda secara otomatis mengelola rute untuk Anda.

PerRegion

Lambda menyediakan nama domain regional terpisah di setiap Wilayah target. Anda bertanggung jawab untuk mengelola bagaimana lalu lintas mencapai setiap domain regional. Misalnya, Anda dapat mengonfigurasi kebijakan perutean Route 53 Anda sendiri atau menempatkan CloudFront distribusi di depan domain regional.

Kegagalan replikasi

Replikasi dapat gagal di Wilayah karena masalah layanan sementara. Jika Anda mengenkripsi Fungsi Web Anda dengan kunci yang dikelola pelanggan, replika kunci yang hilang di Wilayah target juga menyebabkan replikasi gagal. Untuk persyaratan utama Multi-wilayah, lihatthe section called “Mengonfigurasi kunci terkelola pelanggan”.

Efek kegagalan replikasi tergantung pada apakah Anda membuat atau memperbarui titik akhir. Jika replikasi gagal di Wilayah mana pun saat Lambda membuat titik akhir, titik akhir akan berpindah ke status Gagal dan tidak dapat dipanggil di Wilayah mana pun. Untuk mencoba lagi, hapus titik akhir dan buat lagi. Jika replikasi gagal di Wilayah selama pembaruan titik akhir, kegagalan terbatas pada Wilayah tersebut. Titik akhir tetap dapat digunakan, dan Wilayah lainnya terus beroperasi secara normal.

Ketika replikasi revisi baru gagal di Wilayah, Wilayah tersebut terus melayani lalu lintas menggunakan revisi sebelumnya yang berhasil diterapkan. Titik akhir tidak berputar kembali. Ini terus beroperasi dengan negara baik terakhir yang diketahui di setiap Wilayah.

Anda dapat melihat status masing-masing Wilayah dan memperbarui status di `regionalEndpoints` bidang `GetWebFunctionEndpoint` respons. Masing-masing entri Wilayah meliputi:

- `state`dan `stateReason` — Status penyebaran Wilayah selama pembuatan titik akhir.
- `updateStatus`dan `updateStatusReason` — Hasil pembaruan terbaru (misalnya, penerapan revisi) ke Wilayah tersebut.

Untuk mengatasi kegagalan replikasi, atasi akar penyebabnya (misalnya, buat replika AWS KMS kunci di Wilayah yang terpengaruh). Untuk mencoba lagi pembaruan titik akhir yang gagal setelah Anda mengatasi akar penyebabnya, hubungi `UpdateWebFunctionEndpoint` lagi. Anda dapat mengulangi pengaturan yang sama atau hanya menentukan nama fungsi dan nama titik akhir. Salah satu permintaan memulai penerapan baru ke Wilayah target. Ini hanya berfungsi saat titik akhir Aktif. Jika pembuatan titik akhir gagal, hapus titik akhir dan buat lagi.

Konsistensi akhirnya

Replikasi artefak runtime akhirnya konsisten. Jangan berasumsi bahwa setiap Wilayah menjalankan konfigurasi baru hingga setiap Wilayah ada `updateStatus` dalam `regionalEndpoints` laporan`Successful`.

Resource-based perubahan kebijakan dapat selesai menyebar setelah perubahan kode dan konfigurasi. Selama jendela replikasi, permintaan di Wilayah replika mungkin dievaluasi terhadap versi kebijakan sebelumnya. Lihat informasi yang lebih lengkap di the section called “Replikasi dan konsistensi akhirnya”.

Permintaan dan tanggapan HTTP

Endpoint Web Function menerima HTTP/1.1 permintaan dan meneruskannya ke aplikasi Anda. HTTP/1.1 Lambda menolak HTTP/1.0 permintaan dengan HTTP 505 (Versi HTTP Tidak Didukung). Lambda mempertahankan metode permintaan, jalur, string kueri, badan, dan sebagian besar header. Ini menghapus atau mengubah header yang menggambarkan hubungan antara klien dan titik akhir. Ini juga menambahkan header yang menyediakan konteks permintaan.

Metode HTTP

Web Function endpoint mendukung`GET`,`HEAD`,`POST`,`PUT`,`PATCH`,`OPTIONS`, dan`DELETE`.

Lambda menolak metode yang tidak didukung dengan respons HTTP 400 sebelum permintaan mencapai aplikasi Anda.

Minta header

Lambda menghapus header berikut sebelum meneruskan permintaan:

- `Connection`, `Keep-Alive`, `Upgrade`, `Expect`, `TE`, dan `Trailer`
- `Proxy-Authenticate`, `Proxy-Authorization`, dan `Proxy-Connection`
- `X-Real-IP`
- `X-HTTP-Method-Override`, `X-HTTP-Method`, dan `X-Method-Override`
- Header yang namanya dimulai dengan`lambda-web-`, dan header bernama di header `Connection`

Lambda memvalidasi atau mengubah header berikut.

Header	Perilaku
`Host`	Harus berupa nama domain endpoint Web Function (misalnya ,`bcdfghjklmnpqrst.b7t.lambda-web.us-east-1.on.aws`), dalam huruf kecil dan tanpa port. Lambda merutekan nilai ini dan mengembalikan HTTP 404 jika tidak cocok dengan titik akhir, atau HTTP 400 jika header hilang. Aplikasi Anda menerima nama domain endpoint.
`X-Forwarded-For`	Lambda menambahkan alamat IP klien penghubung sebagai nilai terakhir. Nilai sebelumnya berasal dari klien dan tidak divalidasi. Ketika CloudFront distribusi Amazon berada di depan titik akhir, nilai terakhir adalah CloudFront alamat.

Lambda maju `X-Forwarded-Host` dari klien tanpa validasi. Jangan menggunakannya untuk membuat pengalihan, URL absolut, atau otorisasi berbasis host kecuali pintu depan yang Anda kendalikan mengaturnya.

Lambda menambahkan header berikut.

Header	Deskripsi
`lambda-web-request-id`	Pengidentifikasi yang diberikan Lambda ke permintaan.
`lambda-web-endpoint-name`	Nama titik akhir yang menerima permintaan.

Untuk titik akhir yang menggunakan otorisasi IAM, Lambda memvalidasi permintaan yang ditandatangani dan menghapus header sebelum meneruskannya. `Authorization` Untuk informasi selengkapnya, lihat the section called “Menggunakan jenis `IamAuthotentikasi`”.

Header respons

Lambda menghapus header cakupan koneksi dan trailer HTTP dari respons aplikasi Anda. Tergantung pada hasil permintaan, Lambda dapat menambahkan header berikut.

Header	Deskripsi
`lambda-web-request-id`	Pengidentifikasi permintaan. Gunakan untuk mengkorelasikan respons dengan log aplikasi dan sistem.
`lambda-web-logstream`	Aliran CloudWatch log Amazon Logs yang berisi log untuk lingkungan eksekusi yang menangani permintaan.
`lambda-web-error-type`	Jenis kesalahan, dalam format `<Prefix>.<ErrorReason>` (misalnya, `Platform.EndpointRequestThrottle` atau`Runtime.RequestTimeout`). Lambda menghilangkan header ini untuk kesalahan layanan internal dan kegagalan otorisasi. Untuk daftar nilai, lihatthe section called “Jenis kesalahan Fungsi Web”.

Batas header

Baris permintaan, header permintaan, dan mengakhiri baris kosong dapat berjumlah hingga 128 KiB. Lambda menolak permintaan yang melebihi batas ini dengan HTTP 431 (Request Header Fields Too Large) sebelum permintaan mencapai aplikasi Anda. Untuk pemecahan masalah, lihat. the section called “Minta batas header”

Mengkonfigurasi jaringan untuk AWS Lambda Fungsi Web

Fungsi Web menerima permintaan dari klien. Ini juga dapat menjangkau sumber daya lain untuk melakukan pekerjaannya, seperti database di VPC Anda, API internal, atau layanan di internet publik. Anda mengkonfigurasi dua arah ini secara independen. Anda dapat mengekspos titik akhir ke penelepon tanpa memberinya akses VPC, atau memberikan fungsi akses keluar pribadi tanpa mengubah siapa yang dapat memanggilnya.

Lalu lintas masuk mencapai Fungsi Web melalui titik akhir. Endpoint mendefinisikan nama domain, tipe endpoint, Regions, dan konfigurasi otorisasi. Lalu lintas keluar dikonfigurasi pada revisi. Revisi tanpa konektor jaringan jalan keluar mencapai tujuan publik `IPv4` secara langsung. Revisi yang mereferensikan konektor mengirimkan lalu lintas keluar melalui VPC Anda. Fungsi Web berjalan pada infrastruktur yang dimiliki dan dikelola Lambda, bukan di dalam VPC Anda. Konektor jaringan jalan keluar hanya menambahkan akses VPC keluar. Itu tidak mengubah cara klien mencapai titik akhir atau siapa yang diizinkan untuk memanggilnya.

> **Note**
>
> Nama domain endpoint Web Function dapat dijangkau dari internet publik. AWS PrivateLink titik akhir antarmuka tidak tersedia untuk nama domain titik akhir Fungsi Web. Gunakan konfigurasi otorisasi titik akhir untuk mengontrol akses.

Konektivitas Masuk

Klien memanggil Fungsi Web melalui titik akhir HTTPS Lambda-managed publik. Lambda menghentikan TLS dan meneruskan permintaan ke server HTTP aplikasi Anda. Endpoint terkelola mendukung HTTP/1.1. Itu tidak mengekspos alamat VPC pribadi atau menerima koneksi langsung ke lingkungan eksekusi.

`IamAuth`mengontrol siapa yang dapat memanggil titik akhir, tetapi itu tidak membuat titik akhir pribadi di lapisan jaringan. Untuk opsi otorisasi, lihatthe section called “Kontrol akses”.

Endpoint mengontrol permintaan routing, otorisasi, throttling, dan scaling. Jenis titik akhir dan set Wilayah diperbaiki saat Anda membuatnya. Untuk mengubah salah satu properti, buat endpoint baru dan pindahkan klien ke nama domain barunya.

Konfigurasi titik akhir tidak tergantung pada jalan keluar VPC. Menambahkan konektor jaringan keluar ke revisi tidak membuat titik akhir menjadi pribadi dan tidak membatasi permintaan masuk.

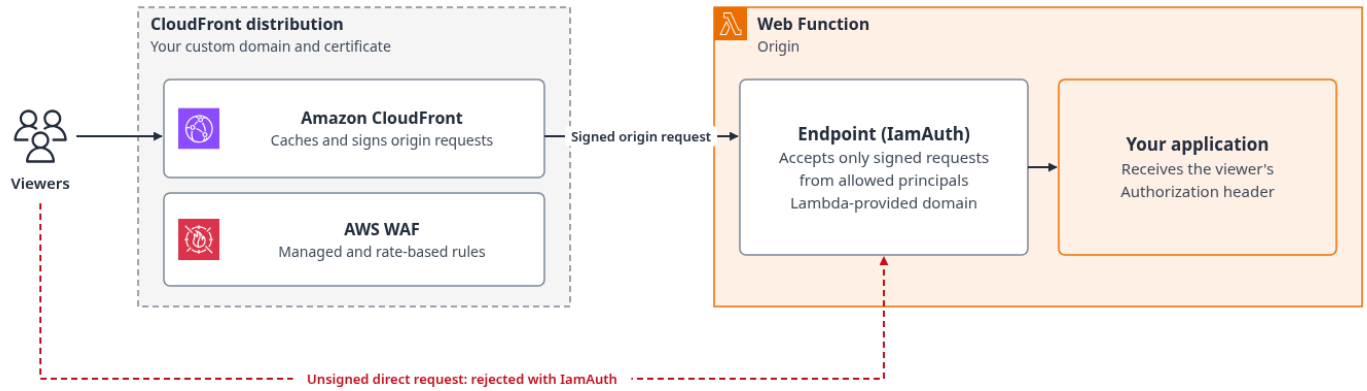
> **Important**
>
> Anda dapat menggunakan konektor jaringan jalan keluar hanya dengan `HomeRegion` titik akhir. Lambda menolak upaya untuk merutekan `PerRegion` titik akhir `MultiRegion` atau revisi yang mereferensikan konektor jaringan keluar.

Ketika bobot revisi membagi lalu lintas antara revisi dengan konfigurasi konektor yang berbeda, permintaan menggunakan jalur jaringan yang berbeda. Saat Anda mendiagnosis konektivitas intermiten, periksa mode penerapan otomatis titik akhir dan bobot revisi terlebih dahulu. Untuk cara mereka merutekan permintaan ke revisi, lihatthe section called “Pola deployment”.

Tambahkan pintu depan HTTP di depan titik akhir

Pintu depan adalah layanan yang menerima permintaan klien dan meneruskannya ke titik akhir Anda. Ini menambahkan kemampuan seperti domain kustom, caching, atau perlindungan firewall aplikasi web. Titik akhir Fungsi Web tidak mendukung nama domain khusus atau AWS WAF asosiasi langsung. Gunakan Amazon CloudFront di depan titik akhir saat Anda memerlukan domain khusus, cache tepi, HTTP/2 atau HTTP/3 antara penampil dan tepi, kompresi respons, atau AWS WAF kontrol.

Diagram berikut menunjukkan pintu depan produksi. Klien terhubung ke domain CloudFront khusus yang dilindungi oleh AWS WAF. CloudFront mengirimkan permintaan ke endpoint Web Function yang dilindungi, yang merutekan ke lingkungan eksekusi. Tanpa konektor jaringan jalan keluar, aplikasi dapat memanggil AWS layanan publik atau titik akhir internet. Itu tidak dapat menjangkau sumber daya yang hanya memiliki alamat VPC pribadi. Untuk mencapai sumber daya tersebut, lihatthe section called “Menghubungkan Fungsi Web ke VPC Anda”. Dengan`IamAuth`, CloudFront tandatangani setiap permintaan asal, dan Lambda mengotorisasi permintaan sebelum menjalankan fungsi Anda. Lambda mempertahankan `Authorization` header asli pemirsa dan meneruskannya ke fungsi Anda. Lambda-provided Domain hanya menerima permintaan yang ditandatangani dari kepala sekolah dengan izin untuk memanggilnya, dan menolak permintaan langsung yang tidak ditandatangani, seperti browser yang memanggil domain secara langsung. Untuk memastikan titik akhir hanya menerima panggilan dari distribusi Anda, tambahkan kebijakan berbasis sumber daya yang memberikan izin pemanggilan ke prinsipal CloudFront layanan dan cakupannya ke Nama Sumber Daya Amazon (ARN) distribusi Anda dengan kondisi ARN sumber. Dengan`ApplicationManaged`, titik akhir bersifat publik, dan permintaan langsung melewati CloudFront dan. AWS WAF



1. Konfigurasikan domain Fungsi Web sebagai asal kustom HTTPS.
2. Lampirkan sertifikat dan nama domain alternatif ke distribusi, dan arahkan catatan Sistem Nama Domain (DNS) Anda ke distribusi.
3. Teruskan hanya header, cookie, dan string kueri yang dibutuhkan aplikasi. Jangan meneruskan `Host` header pemirsa, karena Lambda merutekan `Host` dan mengembalikan HTTP 404 saat itu bukan domain titik akhir. Jika Anda menggunakan kebijakan permintaan asal terkelola yang meneruskan semua header penampil, pilih. `AllViewerExceptHostHeader` Batas header endpoint termasuk header yang diteruskan.
4. (Opsional) Lampirkan ACL AWS WAF web saat Anda membutuhkan inspeksi IP, bot, berbasis tarif, atau aturan terkelola.
5. Lindungi asal sehingga klien tidak dapat melewati distribusi. Untuk IAM-authorized titik akhir, berikan akses hanya ke identitas pintu depan yang dimaksud. Untuk otorisasi yang dikelola aplikasi, mewajibkan dan memvalidasi rahasia asal atau bukti tingkat aplikasi lainnya bahwa permintaan tersebut datang melalui distribusi.

Untuk prosedur CloudFront otorisasi, lihatthe section called “CloudFront integrasi”.

Akses browser dan CORS

Lambda tidak menambahkan header Cross-Origin Resource Sharing (CORS) untuk Fungsi Web. Aplikasi atau pintu depan Anda harus menangani permintaan preflight dan aktual.

- Kembalikan asal yang diizinkan alih-alih mencerminkan setiap `Origin` nilai.
- Izinkan hanya metode dan header permintaan yang digunakan aplikasi.

- Saat browser mengirim kredensyal, gunakan asal dan pengembalian eksplisit. `Access-Control-Allow-Credentials: true` Jangan menggabungkan kredensi dengan wildcard asal.
- Uji permintaan `OPTIONS` preflight secara terpisah dari metode aplikasi.
- Otentikasi dan otorisasi setiap permintaan yang dilindungi. CORS bukan kontrol otentikasi.

Batasan streaming dan protokol

Fungsi Web mengalirkan badan permintaan ke aplikasi Anda saat mereka tiba dan mengalirkan respons kembali ke klien HTTP/1.1. Jalur permintaan setengah dupleks: Lambda meneruskan badan permintaan lengkap sebelum mengirimkan respons aplikasi Anda. Baca badan permintaan sepenuhnya sebelum Anda menulis tanggapan. Setel `timeoutSeconds` untuk mencakup streaming di kedua arah: menerima badan permintaan dan mengirim respons. Saat batas waktu berlalu, Lambda mengakhiri respons. Klien menerima kode status yang sudah dikirim, dengan tubuh terpotong. Hentikan pekerjaan upstream saat klien terputus.

Lambda tidak mendukung WebSocket peningkatan. Lambda menghapus header pemutakhiran khusus koneksi sebelum meneruskan permintaan. Gunakan acara yang dikirim server saat streaming server-ke-klien satu arah sudah cukup.

Untuk kontrak header dan metode lengkap, lihatthe section called “Perilaku HTTP”.

Konektivitas keluar

Anda memilih jalur jaringan keluar untuk setiap revisi.

- Tanpa konektor jaringan jalan keluar — Revisi mencapai `IPv4` tujuan publik secara langsung tanpa melintasi VPC Anda. Alamat sumbernya Lambda-managed, tidak stabil, dan tidak cocok untuk daftar izin. Revisi tidak memiliki rute ke sumber daya pribadi di VPC Anda.
- Dengan konektor jaringan keluar - Revisi mengirimkan lalu lintas keluar melalui antarmuka jaringan elastis di subnet konektor. Tabel rute VPC Anda, grup keamanan, dan daftar kontrol akses jaringan (ACL) berlaku. Gunakan opsi ini untuk menjangkau sumber daya VPC pribadi, jaringan lokal yang terhubung melalui Direct Connect atau VPN, atau tujuan yang memerlukan lalu lintas dari ruang alamat Anda.

Mereferensikan konektor menggantikan jalur keluar default. Konektor menempatkan lalu lintas di VPC Anda, tetapi tidak menyediakan akses internet. Untuk mencapai `IPv4` tujuan umum, subnet konektor

memerlukan rute ke gateway NAT. Untuk mencapai IPv6 tujuan umum, mereka membutuhkan rute ke gateway internet khusus egres.

Tidak ada alamat IP sumber yang stabil untuk lalu lintas konektor. Konfigurasikan grup keamanan tujuan untuk mengizinkan grup keamanan konektor. Jika tujuan tidak mendukung referensi grup keamanan, izinkan blok CIDR subnet konektor alih-alih alamat individual.

Lambda menjalankan operasi konektor secara asinkron. Sebuah konektor melaporkan PENDING sementara Lambda membuat antarmuka jaringannya, dan ACTIVE ketika revisi dapat mereferensikannya. Negara-negara lain termasuk INACTIVEFAILED,DELETING,, danDELETE_FAILED.

Setelah pembaruan, periksa LastUpdateStatusInProgress,Successful, atauFailed. Jika operasi gagal, gunakan alasan status dan kode alasannya untuk mengidentifikasi subnet, grup keamanan, izin, atau kapasitas alamat IP yang terpengaruh.

Resolusi DNS

Untuk nama DNS pribadi, aktifkan dukungan DNS dan nama host DNS di VPC konektor. Kaitkan zona host pribadi Route 53 yang diperlukan dengan VPC tersebut.

Setiap lingkungan eksekusi menyimpan jawaban DNS. Setelah mengubah catatan DNS, lingkungan yang ada terus menggunakan jawaban lama hingga nilai cache kedaluwarsa atau Lambda menggantikan lingkungan. Untuk memaksa peluncuran yang bersih, buat revisi baru dan geser lalu lintas titik akhir ke sana.

Saat memecahkan masalah, uji resolusi nama dan jangkauan jaringan secara terpisah. Connect ke tujuan dengan alamat IP untuk memverifikasi jalur jaringan, lalu uji nama DNS-nya.

Menghubungkan Fungsi Web ke VPC Anda

Konektor jaringan jalan keluar adalah sumber daya Lambda yang dapat digunakan kembali. Anda membuat konektor sekali dan mereferensikan versi konektor yang diterbitkan dari setiap revisi Fungsi Web yang memerlukan akses VPC.

Untuk Fungsi Web, referensi konektor memiliki properti berikut:

- Revisi referensi nol atau satu konektor.
- Konektor ARN harus menyertakan kualifikasi versi.
- Konektor harus berada di AWS akun dan Wilayah yang sama dengan Fungsi Web.

- Konektor harus memiliki `WebFunction` jenis sumber daya komputasi yang terkait.
- Hanya `HomeRegion` titik akhir yang mendukung revisi yang mereferensikan konektor jaringan keluar.
- Anda tidak dapat mengubah referensi setelah Anda mengaturnya. Untuk menambah, mengubah, atau menghapusnya, buat revisi baru.

Prasyarat

Sebelum Anda membuat konektor:

- Pilih setidaknya dua subnet di Availability Zone yang berbeda dengan alamat IP yang tersedia.
- Pilih grup keamanan yang memungkinkan lalu lintas keluar yang diperlukan.
- Izinkan lalu lintas masuk dari grup keamanan konektor di setiap tujuan.
- Konfigurasikan rute dari subnet konektor ke tujuan pribadi, lokal, atau publik yang diperlukan.
- Buat peran operator yang memberikan izin Lambda untuk mengelola antarmuka jaringan.
- Pastikan identitas yang membuat konektor dan revisi memiliki izinthe section called “Izin untuk mengkonfigurasi jaringan”, termasuk `lambda:PassNetworkConnector` pada konektor dan peran `iam:PassRole` operator.

Peran operator konektor dan peran eksekusi revisi melayani tujuan yang berbeda. Peran operator mengelola antarmuka jaringan di VPC Anda. Peran eksekusi memberikan izin aplikasi Anda untuk memanggil AWS API. Anda tidak menambahkan izin antarmuka jaringan konektor ke peran eksekusi.

Izin peran operator

Lambda mengasumsikan peran operator konektor untuk mengelola antarmuka jaringan konektor. Kebijakan kepercayaan peran harus memungkinkan kepala layanan Lambda,`lambda.amazonaws.com`, untuk menganggapnya:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "lambda.amazonaws.com"
      },
```

```
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Lampirkan kebijakan izin seperti berikut ini ke peran operator konektor. Kebijakan ini memungkinkan Lambda untuk membuat dan menandai antarmuka jaringan konektor:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "CreateENI",
      "Effect": "Allow",
      "Action": "ec2:CreateNetworkInterface",
      "Resource": [
        "arn:aws:ec2:*:*:network-interface/*",
        "arn:aws:ec2:*:*:subnet/*",
        "arn:aws:ec2:*:*:security-group/*"
      ]
    },
    {
      "Sid": "TagENI",
      "Effect": "Allow",
      "Action": "ec2:CreateTags",
      "Resource": "arn:aws:ec2:*:*:network-interface/*",
      "Condition": {
        "StringEquals": {
          "ec2:ManagedResourceOperator": "network-connectors.lambda.amazonaws.com"
        }
      }
    }
  ]
}
```

Membuat konektor

AWS CLI Perintah berikut membuat `IPv4` konektor untuk Web Functions. Ganti nama konektor, ID subnet, ID grup keamanan, dan peran operator ARN dengan nilai Anda sendiri.

```
aws lambda-core create-network-connector \
  --name my-connector \
```

```
  --configuration '{
    "VpcEgressConfiguration": {
      "SubnetIds": [
        "subnet-0ab1c2d3e4f5a6b78",
        "subnet-1bc2d3e4f5a6b7c89"
      ],
      "SecurityGroupIds": ["sg-0123456789abcdef0"],
      "NetworkProtocol": "IPv4",
      "AssociatedComputeResourceTypes": ["WebFunction"]
    }
  }' \
  --operator-role arn:aws:iam::123456789012:role/NetworkConnectorOperatorRole
```

Konektor mendukung satu jenis sumber daya komputasi. Konektor untuk Fungsi Web tidak mendukung jenis komputasi Lambda lain, dan jenis sumber daya komputasinya tidak dapat diubah.

Lambda membuat konektor secara asinkron. Gunakan `get-network-connector` untuk menunggu sampai `State` ada`ACTIVE`, dan catat nomor versi saat ini.

Referensi konektor dari revisi

Setel `serviceConfig.egressNetworkConnectorArn` saat Anda membuat revisi pertama Fungsi Web atau revisi selanjutnya. Gunakan ARN yang memenuhi syarat versi:

```
arn:aws:lambda:us-east-1:123456789012:network-connector:nc-0a1b2c3d4e5f6a7b8:4
```

AWS CLI Perintah berikut membuat revisi yang referensi konektor versi 4. Ganti nama fungsi, peran eksekusi ARN, konektor ARN, dan Wilayah dengan nilai Anda sendiri.

```
aws lambda-web create-web-function-revision \
  --function-name my-web-function \
  --build-config file://build-config.json \
  --service-config '{
    "executionRoleArn": "arn:aws:iam::123456789012:role/WebFunctionExecutionRole",
    "timeoutSeconds": 30,
    "egressNetworkConnectorArn": "arn:aws:lambda:us-east-1:123456789012:network-
connector:nc-0a1b2c3d4e5f6a7b8:4"
  }' \
  --region us-east-1
```

ARN, nama konektor, atau ID konektor yang tidak memenuhi syarat tidak diterima pada revisi. Setelah membuat revisi:

- Tunggu sampai keadaan revisi. `Active`
- Baca revisi dan konfirmasikan yang `egressNetworkConnectorArn` berisi versi konektor yang diharapkan.
- Konfirmasikan bahwa titik akhir mengarahkan lalu lintas ke revisi. `HomeRegion`Titik akhir dalam `LatestRevision` mode mengambil revisi aktif terbaru secara otomatis. Untuk titik akhir dalam `Disabled` mode, perbarui bobot revisinya.
- Uji tujuan VPC dan konfirmasikan lalu lintas dengan VPC Flow Logs.

Mode penerapan otomatis titik akhir menentukan apakah membuat revisi aktif memperbarui perutean secara otomatis. Di semua mode, konfirmasikan revisi mana yang disajikan sebelum menguji konektornya.

Memperbarui atau menghapus konektivitas VPC

Memperbarui konektor menerbitkan versi baru. Revisi yang ada tetap disematkan ke versi di konektor ARN mereka.

Untuk meluncurkan perubahan konektor:

- Perbarui konektor dengan`update-network-connector`. Konektor harus`ACTIVE`. Berikan lengkap`VpcEgressConfiguration`, termasuk setiap ID subnet dan ID grup keamanan yang ingin Anda simpan, karena pembaruan menggantikan konfigurasi yang ada.
- Tunggu konektornya `LastUpdateStatus` menjadi`Successful`.
- Dapatkan versi konektor baru.
- Buat revisi yang mereferensikan ARN yang memenuhi syarat versi baru.
- Untuk `HomeRegion` titik akhir dalam `LatestRevision` mode, konfirmasikan bahwa itu mengambil revisi aktif baru. Untuk titik akhir dalam `Disabled` mode, geser lalu lintas dengan memperbarui bobot revisinya.

Memperbarui konektor tanpa membuat revisi yang mereferensikan versi barunya tidak mengubah konfigurasi jaringan menjalankan Fungsi Web.

Untuk menghapus konektivitas VPC, buat revisi tanpa. `egressNetworkConnectorArn` `HomeRegion`Titik akhir dalam `LatestRevision` mode mengambilnya saat menjadi aktif. Untuk titik akhir dalam `Disabled` mode, geser lalu lintas dengan memperbarui bobot revisinya. Revisi baru menggunakan jalan `IPv4` keluar Lambda-managed publik dan tidak lagi memiliki akses VPC pribadi.

> **Note**
>
> Sebelum menghapus konektor jaringan keluar, alihkan lalu lintas titik akhir dari setiap revisi Fungsi Web yang mereferensikan versi konektor apa pun, lalu hapus revisi tersebut. Menghapus konektor saat revisi aktif masih mereferensikannya mengganggu konektivitas jaringan untuk lingkungan eksekusi yang ada dan baru yang melayani revisi itu.

Dukungan IPv6

Jalur keluar default dan `IPv4` konektor `IPv4` hanya mendukung. Untuk menggunakan`IPv6`, konfigurasikan konektor dengan `NetworkProtocol` set ke`DualStack`.

Untuk konektivitas dual-stack:

- Subnet VPC dan konektor harus memiliki `IPv6` blok CIDR.
- Tabel rute harus berisi rute untuk `IPv6` tujuan yang diperlukan.
- Grup keamanan dan ACL jaringan harus mengizinkan `IPv6` lalu lintas.
- `IPv6`Akses publik memerlukan rute: :/0 ke gateway internet khusus egress.
- Referensikan versi konektor yang dibuat saat Anda memperbarui konektor ke`DualStack`.

Jangan menentukan `IPv6` ketersediaan dengan memeriksa antarmuka di dalam lingkungan eksekusi. Verifikasi dengan membuat `IPv6` koneksi dari kode aplikasi dan memeriksa VPC Flow Logs.

Perilaku penskalaan untuk Lambda Web Functions

Lambda menskalakan titik akhir Fungsi Web Anda secara otomatis saat lalu lintas tiba. Ini berskala dalam dua dimensi:

- Jumlah lingkungan - Lambda menambahkan lebih banyak lingkungan eksekusi, hingga `maxEnvironments`.
- Permintaan bersamaan per lingkungan — Setiap lingkungan menangani beberapa permintaan sekaligus. Lambda mengelola batas konkurensi dinamis per lingkungan berdasarkan penggunaan sumber daya Anda. Anda mengontrol batas atas batas dinamis ini menggunakan `maxConcurrencyPerEnvironment`. Untuk informasi selengkapnya, lihat the section called “Batas konkurensi dinamis per lingkungan”.

Anda juga dapat mengatur `rateLimit` pada titik akhir untuk membatasi tingkat kedatangan permintaannya. Ini adalah pengaturan opsional dan diterapkan sebelum permintaan mencapai lingkungan Anda.

Titik akhir dinaikkan dari nol. Ketika tidak ada lalu lintas, Lambda tidak menjalankan lingkungan. Halaman ini menjelaskan pengaturan penskalaan titik akhir, kuota akun terkait, dan cara memilih nilai yang mengoptimalkan biaya atau kinerja.

Lingkungan eksekusi

Lingkungan eksekusi adalah instance komputasi terisolasi yang menjalankan server HTTP Anda. Setiap lingkungan memiliki memori, sistem file, dan ruang prosesnya sendiri. Untuk cara Lambda menginisialisasi, menggunakan kembali, dan mematikan lingkungan eksekusi, lihat the section called “Siklus hidup lingkungan”.

Lambda membuat dan menghapus lingkungan secara otomatis berdasarkan lalu lintas:

- Ketika lalu lintas meningkat, Lambda menciptakan lingkungan tambahan untuk memenuhi permintaan lalu lintas.
- Ketika lalu lintas berkurang, Lambda menguras lingkungan yang kurang dimanfaatkan dan menghilangkannya.
- Ketika tidak ada lalu lintas, Lambda menghapus semua lingkungan. Titik akhir berskala menjadi nol.

Tidak ada pengaturan lingkungan minimum atau cadangan. Titik akhir selalu berskala dari nol saat lalu lintas tiba. Untuk menghindari cold start untuk peningkatan lalu lintas kecil, Lambda secara otomatis meningkatkan lingkungan eksekusi sebelum permintaan untuk mempertahankan kapasitas cadangan (headroom) untuk lalu lintas masuk. Ketika lingkungan kurang dimanfaatkan, Lambda menurunkannya, termasuk menjadi nol.

Batas konkurensi dinamis per lingkungan

Batas konkurensi dinamis per lingkungan adalah a Lambda-managed membatasi jumlah permintaan yang ditangani oleh lingkungan eksekusi tunggal pada saat yang sama. Berdasarkan penggunaan sumber daya fungsi Anda, Lambda secara otomatis menyesuaikan batas ini untuk mengoptimalkan efisiensi dan kinerja. Lambda meningkatkan lingkungan baru ketika lingkungan yang ada mendekati batasnya. Ketika batas mencegah semua lingkungan yang ada menerima lebih banyak permintaan, antrian permintaan, menunggu lingkungan yang tersedia berikutnya.

Untuk Fungsi Web baru, Lambda menetapkan batas awal konservatif dan dengan cepat menyesuaikannya berdasarkan penggunaan sumber daya fungsi. Untuk informasi selengkapnya tentang bagaimana batas berubah saat menangani beban, lihat the section called “Batas konkurensi dinamis dari Fungsi Web baru”.

Anda tidak mengonfigurasi batas dinamis secara langsung. Sebagai gantinya, Anda dapat menerapkan batas atas untuk itu dengan mengatur `maxConcurrencyPerEnvironment` pada revisi. Batas dinamis Lambda bisa lebih rendah dari nilai ini, tetapi tidak pernah melebihi itu. Misalnya, jika Anda mengatur `maxConcurrencyPerEnvironment` hingga 32, Lambda merutekan paling banyak 32 permintaan bersamaan ke lingkungan apa pun, bahkan jika lingkungan memiliki kapasitas untuk lebih.

Sebagian besar aplikasi dapat mengandalkan batas dinamis tanpa berubah `maxConcurrencyPerEnvironment`. Untuk panduan tentang kapan harus mengubahnya, lihat the section called “Memilih pengaturan penskalaan”.

Pengaturan penskalaan

Tiga pengaturan mengontrol bagaimana Lambda menskalakan Fungsi Web Anda. Anda mengonfigurasi setiap pengaturan pada titik akhir atau revisi.

Pengaturan	Dikonfigurasi pada	Rentang yang valid	Default	Deskripsi
`maxConcurrencyPerEnvironment`	Revisi	1—128	64	Batas atas Lambda-managed batas konkurensi dinamis per lingkungan. Lambda menyesuaikan konkurensi aktual per lingkungan secara otomatis, sehingga sebagian besar aplikasi dapat mengandalkan batas dinamis tanpa mengubah nilai ini.
`maxEnvironments`	Titik akhir	Tidak wajib. 2—10.000	Tidak ada	Membatasi jumlah lingkungan eksekusi untuk titik akhir di masing-masing Wilayah AWS.

Pengaturan	Dikonfigurasi pada	Rentang yang valid	Default	Deskripsi
				Mengontrol skala horizontal. Penskalaan lingkungan titik akhir selalu dibatasi oleh kuota akun vCPU yang tersedia, bahkan jika `maxEnvironments` tidak diatur.
`rateLimit`	Titik akhir	Tidak wajib. 0—10.000	Tidak ada	Membatasi tingkat kedatangan permintaan titik akhir, dalam permintaan per detik, di setiap Wilayah AWS. Nilai hingga 1.000 harus kelipatan 100, dan nilai di atas 1.000 harus kelipatan 1.000. Nilai 0 memblokir semua lalu lintas. Permintaan di atas batas tarif efektif menerima respons HTTP 429 (Terlalu Banyak Permintaan). Jika tidak disetel, tingkat kedatangan titik akhir dibatasi oleh kuota tarif akun per titik akhir.

`maxEnvironments` dan `rateLimit` adalah properti titik akhir opsional. Anda dapat memperbaruinya tanpa membuat revisi baru. Secara default, keduanya tidak disetel. Sebuah besar `maxEnvironments` tidak menjamin bahwa titik akhir dapat menskalakan sejauh itu. Penskalaan lingkungan selalu dibatasi oleh kuota vCPU regional akun. SEBUAH `rateLimit` dari 0 memblokir semua lalu lintas. Lihat perinciannya di the section called “Throttling”. Tingkat permintaan titik akhir selalu dibatasi oleh kuota tarif permintaan akun, bahkan ketika `rateLimit` tidak diatur. Untuk kuota, lihat bagian selanjutnya.

Untuk berubah `maxEnvironments` atau `rateLimit`, lari `aws lambda-web update-web-function-endpoint` dengan `--scaling-config` atau `--throttle-config` parameter,

misalnya `--scaling-config '{"maxEnvironments": 50}'` atau `--throttle-config '{"rateLimit": 8000}'`. Menghilangkan parameter untuk mempertahankan nilainya saat ini. Untuk menghapus nilai yang Anda tetapkan, berikan objek kosong, seperti `--scaling-config '{}'`. Untuk informasi lebih lanjut, lihat the section called “Memperbarui titik akhir”. `maxConcurrencyPerEnvironment` adalah pengaturan revisi. Untuk mengubahnya, buat revisi baru. Apakah revisi baru menerima lalu lintas tergantung pada mode penyebaran otomatis titik akhir. Untuk informasi selengkapnya, lihat the section called “Membuat revisi”.

Kuota akun

Kuota ini berlaku per Wilayah AWS. Mereka selalu berlaku selain batas titik akhir yang dikonfigurasi secara opsional.

- Per-endpoint batas kecepatan — 10.000 permintaan per detik (RPS).
- Batas tarif total akun — 100.000 RPS di semua titik akhir di Wilayah.
- Batas akun vCPU — 2.000 vCPU di semua titik akhir di Wilayah. Setiap lingkungan eksekusi mengkonsumsi 2 vCPU, jadi kuota ini membatasi jumlah total lingkungan per Wilayah AWS di seluruh akun.

Untuk nilai kuota saat ini dan cara meminta kenaikan, lihat the section called “Fungsi Web Lambda”.

Bagaimana pengaturan penskalaan berinteraksi

Ketiga pengaturan penskalaan ini beroperasi secara independen untuk menentukan skala titik akhir Anda. Kecuali untuk kuota akun, setiap titik akhir dinilai secara independen, bahkan titik akhir milik Fungsi Web yang sama.

Total kapasitas konkurensi

> Anda dapat menganggap jumlah puncak permintaan dalam penerbangan titik akhir sebagai kira-kira `maxEnvironments` dikalikan batas konkurensi dinamis per lingkungan yang saat ini diizinkan Lambda. Yang paling bisa ditangani titik akhir sekaligus adalah `maxEnvironments` × `maxConcurrencyPerEnvironment`. Itu mencapai angka itu hanya ketika berjalan `maxEnvironments` lingkungan dan masing-masing menangani `maxConcurrencyPerEnvironment` permintaan. Jika `maxEnvironments` tidak disetel, titik akhir dinaikkan untuk memenuhi permintaan, hingga kuota akun vCPU.

Batas tarif permintaan total

`rateLimit` membatasi permintaan per detik, terlepas dari kapasitas konkurensi. Lambda mengembalikan HTTP 429 untuk permintaan di atas batas tarif, bahkan ketika lingkungan memiliki kapasitas untuk melayani mereka.

Meningkatkan

Sebelum semua lingkungan mencapai batas konkurensi dinamis, Lambda meningkatkan lebih banyak lingkungan, hingga `maxEnvironments`. Titik akhir berhenti meningkatkan skala saat mencapai `maxEnvironments` atau kuota akun vCPU. Jika titik akhir tidak dapat ditingkatkan, dan semua lingkungan yang ada dimanfaatkan sepenuhnya, Lambda segera membatasi permintaan baru dengan respons HTTP 429 (Too Many Requests). Lambda tidak mengantri permintaan yang dibatasi. Jika lingkungan eksekusi baru diinisialisasi, meminta antrian alih-alih segera dibatasi.

Overshoot sementara

Selama perubahan bobot revisi, siklus armada internal, dan peristiwa operasional lainnya, Lambda mungkin untuk sementara melebihi `maxEnvironments` untuk memprioritaskan ketersediaan. Overshoot ini singkat, dan Lambda kembali ke batas yang dikonfigurasi secara otomatis. Anda tidak perlu menganggarkan ruang untuk acara operasional ini ketika Anda memilih `maxEnvironments`.

Example Contoh: perhitungan kapasitas

Anda mengonfigurasi pengaturan berikut untuk Fungsi Web Anda:

- `maxEnvironments` = 10
- `maxConcurrencyPerEnvironment` = 64
- `rateLimit` = 8000

Pengaturan ini membatasi konkurensi (permintaan dalam penerbangan sekaligus) dan tarif permintaan (permintaan diterima per detik) secara terpisah. Dengan hingga 10 lingkungan dan masing-masing hingga 64 permintaan bersamaan, titik akhir dapat melayani paling banyak sekitar 640 permintaan bersamaan (10 × 64). Batas dinamis seringkali lebih rendah, sehingga konkurensi berkelanjutan biasanya kurang. Batas mana yang dicapai titik akhir pertama tergantung pada berapa lama waktu yang dibutuhkan setiap permintaan.

Concurrency-bound titik akhir

Misalkan setiap permintaan membutuhkan waktu 200 ms, misalnya karena menunggu pada query database. Dengan 640 permintaan dalam penerbangan, titik akhir menyelesaikan paling banyak sekitar 3.200 permintaan per detik (640 ÷ 0,2 detik), yang di bawah 8.000 RPS `rateLimit`. Ketika semua 10 lingkungan dimanfaatkan sepenuhnya, Lambda membatasi permintaan baru dengan HTTP 429. Untuk melayani lebih banyak lalu lintas, tingkatkan `maxEnvironments`.

Request-rate-bound titik akhir

Misalkan setiap permintaan membutuhkan waktu 10 ms. Pada 8.000 permintaan per detik, sekitar 80 permintaan sedang dalam penerbangan (8.000 × 0,01 detik), jauh di bawah 640. Lambda mengembalikan HTTP 429 untuk permintaan di atas 8.000 RPS, meskipun lingkungan memiliki kapasitas untuk melayani mereka. Untuk menerima lebih banyak permintaan, tingkatkan `rateLimit`, hingga kuota 10.000 RPS per titik akhir.

Throttling

Ketika tingkat permintaan masuk melebihi `rateLimit` pengaturan, Lambda mengembalikan respons HTTP 429 (Terlalu Banyak Permintaan) untuk permintaan berlebih. Batas tarif berlaku untuk titik akhir secara keseluruhan di masing-masing Wilayah AWS.

Lambda juga mengembalikan respons HTTP 429 ketika semua lingkungan dimanfaatkan sepenuhnya, baik karena titik akhir tercapai `maxEnvironments` atau akun mencapai kuota vCPU-nya. Untuk informasi selengkapnya, lihat the section called “Bagaimana pengaturan penskalaan berinteraksi”.

Lambda juga dapat mengembalikan HTTP 429 dengan `Platform.EndpointConcurrentInitThrottle` jenis kesalahan ketika terlalu banyak lingkungan eksekusi yang diinisialisasi pada saat yang sama. Throttle ini tidak menambah metrik throttle. Untuk metrik yang dinaikkan oleh jenis throttle lainnya, lihat the section called “Metrik-metrik”.

Ketika titik akhir `rateLimit` adalah 0, Lambda mengembalikan HTTP 503 (Layanan Tidak Tersedia) alih-alih HTTP 429, dengan `lambda-web-error-type` header `Platform.EndpointDisabled`. Lambda juga dapat mengembalikan HTTP 503 ketika melepaskan beban untuk melindungi layanan. Coba lagi 503 yang tidak memiliki `lambda-web-error-type` header dengan backoff dan jitter. A 503 dengan `Platform.EndpointDisabled` berlanjut sampai Anda mengatur `rateLimit` ke nilai bukan nol.

Lambda mengukur batas laju per detik. Semburan singkat di atas batas mungkin diterima. Lalu lintas berkelanjutan di atas batas menghasilkan pelambatan.

Saat aplikasi Anda menerima tanggapan HTTP 429, terapkan backoff eksponensial dengan jitter dalam kode klien Anda untuk mencoba lagi permintaan yang dibatasi.

Menaikkan dan menurunkan skala

Penaikan skala

Ketika lalu lintas meningkat dan lingkungan yang ada mendekati batas konkurensi dinamis per lingkungan, Lambda menciptakan lingkungan eksekusi tambahan. Setiap lingkungan baru diinisialisasi sebelum melayani permintaan. Untuk langkah-langkah inisialisasi dan batas waktu startup, lihat the section called “Inisialisasi (start dingin)”.

Sementara lingkungan baru diinisialisasi, Lambda merutekan permintaan ke lingkungan yang ada yang memiliki kapasitas tersedia. Ketika tidak ada lingkungan yang memiliki kapasitas, permintaan menunggu sementara Lambda menginisialisasi yang baru (start dingin). Lambda mengembalikan HTTP 429 hanya ketika salah satu batas di the section called “Throttling” berlaku, seperti `maxEnvironments` atau kuota akun vCPU.

Menurunkan skala

Ketika lalu lintas berkurang, Lambda mengidentifikasi lingkungan yang kurang dimanfaatkan dan menghapusnya. Lambda menguras lingkungan sebelum menghapusnya. Ini berhenti merutekan permintaan baru ke lingkungan, memungkinkan permintaan dalam penerbangan diselesaikan dengan upaya terbaik, dan mengirim `SIGTERM`. Lihat perinciannya di the section called “Siklus hidup lingkungan”.

Lambda menskalakan ke lingkungan nol ketika tidak ada lalu lintas. Permintaan berikutnya setelah itu memulai lingkungan baru (start dingin).

Memilih pengaturan penskalaan

Pertimbangkan hal berikut saat Anda memilih nilai untuk titik akhir `rateLimit`, titik akhir `maxEnvironments`, dan revisi `maxConcurrencyPerEnvironment`.

Memilih `rateLimit`

Nilai defaultnya tidak ada, yang berarti tingkat permintaan titik akhir dibatasi oleh kuota RPS 10.000 per titik akhir dan kuota RPS 100.000 akun. Pertimbangkan untuk mengatur a `rateLimit`

nilai ketika Anda ingin membatasi berapa banyak titik akhir yang dapat diklaim dari kuota RPS akun Anda, yang dibagikan oleh semua titik akhir Anda.

Mengatur a `rateLimit` mungkin juga membantu untuk membatasi lalu lintas total ke dependensi hilir.

Memilih `maxEnvironments`

Nilai defaultnya tidak ada, yang berarti jumlah lingkungan titik akhir hanya dibatasi oleh kuota vCPU akun. Pertimbangkan untuk mengatur a `maxEnvironments` nilai ketika Anda ingin membatasi berapa banyak titik akhir yang dapat diklaim dari kuota vCPU akun Anda, yang dibagikan oleh semua titik akhir Anda.

Saat mengatur a `maxEnvironments` nilai, pertimbangkan untuk memantau jumlah lingkungan titik akhir sehingga Anda diperingatkan ketika dekat dengan `maxEnvironments` membatasi, dan mengambil tindakan untuk menghindari pelambatan untuk aplikasi yang memiliki pertumbuhan organik dari waktu ke waktu.

Menetapkan nilai rendah dari `maxEnvironments` dapat membantu membatasi biaya menjalankan lingkungan untuk titik akhir Anda. Pertimbangkan untuk menggunakan nilai rendah untuk proyek hobi non-kritis.

Memilih `maxConcurrencyPerEnvironment`

Sebuah revisi `maxConcurrencyPerEnvironment` adalah batas atas batas konkurensi dinamis per lingkungan yang dikelola Lambda. Nilai defaultnya adalah 64, dan sebagian besar pengguna tidak harus memilih nilai optimal.

Untuk aplikasi yang terikat memori, pengaturan `maxConcurrencyPerEnvironment` lebih rendah dari default dapat membantu mencapai perilaku yang lebih dapat diprediksi.

Untuk CPU-bound aplikasi, pengaturan `maxConcurrencyPerEnvironment` lebih rendah dari default dapat mengurangi perselisihan CPU dan meningkatkan latensi keadaan stabil di lingkungan yang hangat, tetapi juga dapat mengakibatkan start dingin yang lebih sering. Pengaturan `maxConcurrencyPerEnvironment` lebih tinggi dari default kemungkinan tidak akan berpengaruh jika batas konkurensi dinamis terkelola Lambda membatasi lalu lintas ke lingkungan.

Untuk I/O-bound aplikasi, pengaturan `maxConcurrencyPerEnvironment` lebih tinggi dari default dapat meningkatkan pemanfaatan per lingkungan dan biaya yang lebih rendah.

Pantau yang berikut CloudWatch metrik untuk menentukan apakah pengaturan penskalaan Anda sesuai. Untuk definisi metrik, lihat the section called “Metrik-metrik”.

- `EndpointRequestThrottles` dan `AccountRequestThrottles` — Jika `EndpointRequestThrottles` secara konsisten lebih besar dari nol, meningkat `rateLimit`, hingga kuota per titik akhir 10.000 RPS. `AccountRequestThrottles` meningkat saat akun mencapai kuota tingkat permintaannya di Wilayah. Untuk mengurangi throttle ini, minta kenaikan kuota, atau turunkan `rateLimit` pada titik akhir lainnya.
- `EndpointActiveEnvironmentThrottles` dan `AccountActiveVCPUArm64Throttles` — Metrik ini bertambah ketika `maxEnvironments` atau kuota akun vCPU mencegah penskalaan tambahan. Untuk mengurangi throttle ini, tingkatkan `maxEnvironments` atau minta peningkatan kuota vCPU.
- `EndpointActiveEnvironments` dan `EnvironmentConcurrentRequests` — Jika lingkungan aktif mendekati `maxEnvironments`, atau permintaan bersamaan per lingkungan tetap dekat `maxConcurrencyPerEnvironment`, tingkatkan pengaturan yang relevan.
- Latensi aplikasi — Ukur latensi dengan metrik aplikasi, dari log terstruktur, atau dengan pengukuran latensi pintu depan Anda. Jika latensi meningkat seiring meningkatnya konkurensi per lingkungan, kurangi `maxConcurrencyPerEnvironment` untuk mengurangi perselisihan per lingkungan dan menyebarkan beban di lebih banyak lingkungan.

Penskalaan untuk MultiRegion dan PerRegion titik akhir

Untuk Fungsi Web dengan MultiRegion atau PerRegion titik akhir, pengaturan penskalaan berlaku secara independen di masing-masing Wilayah AWS di mana titik akhir beroperasi.

Setiap Wilayah menciptakan dan mengelola kumpulan lingkungan eksekusi sendiri. Lalu lintas di satu Wilayah tidak mengkonsumsi kapasitas penskalaan dari Wilayah lain.

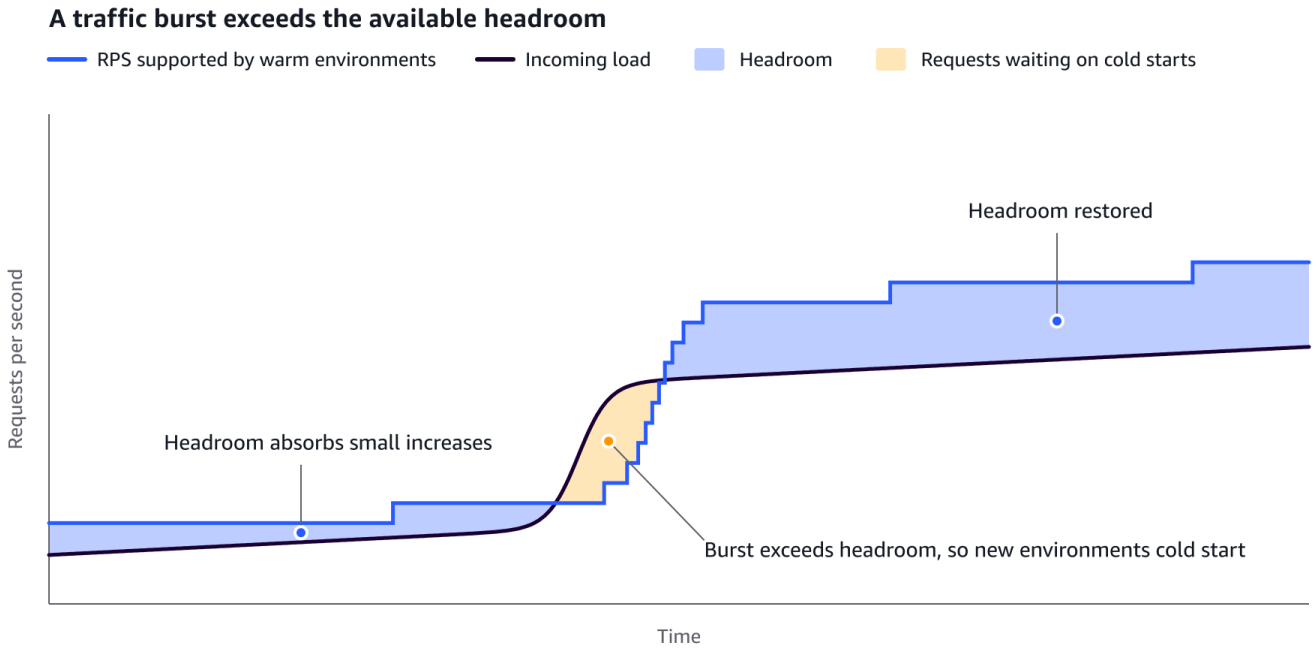
Contoh penskalaan

Lambda menskalakan titik akhir Anda untuk mempertahankan ruang kepala di atas beban saat ini. Headroom adalah perbedaan antara beban permintaan masuk dan beban permintaan total yang dapat dilayani oleh lingkungan eksekusi hangat. Ini menyerap peningkatan lalu lintas kecil tanpa menunggu lingkungan baru dimulai. Kapasitas meningkat secara bertahap, karena setiap lingkungan baru menambah kapasitas hanya setelah selesai menginisialisasi. Ketika titik akhir mencapai batas, kapasitas berhenti tumbuh, dan Lambda membatasi permintaan yang melebihi apa yang dapat diterima titik akhir.

Contoh berikut menunjukkan bagaimana titik akhir menangani ledakan lalu lintas, bagaimana batas konkurensi dinamis per lingkungan berubah selama pengujian beban pada Fungsi Web baru, dan bagaimana titik akhir berperilaku ketika beban mencapai setiap jenis batas. Bagan menggambarkan perilaku umum dan tidak digambar sesuai skala.

Menangani ledakan lalu lintas

Ketika beban tumbuh secara bertahap, headroom menyerap peningkatan sementara Lambda meningkatkan lingkungan baru di latar belakang. Ketika beban naik lebih cepat daripada yang dapat diserap ruang kepala, lingkungan yang hangat tidak dapat menangani semua permintaan yang masuk. Lambda memulai lingkungan baru untuk menangani beban di atas headroom, dan permintaan tersebut menunggu lingkungan baru mulai dingin. Setelah lingkungan baru siap, Lambda mengembalikan ruang kepala di atas beban baru.

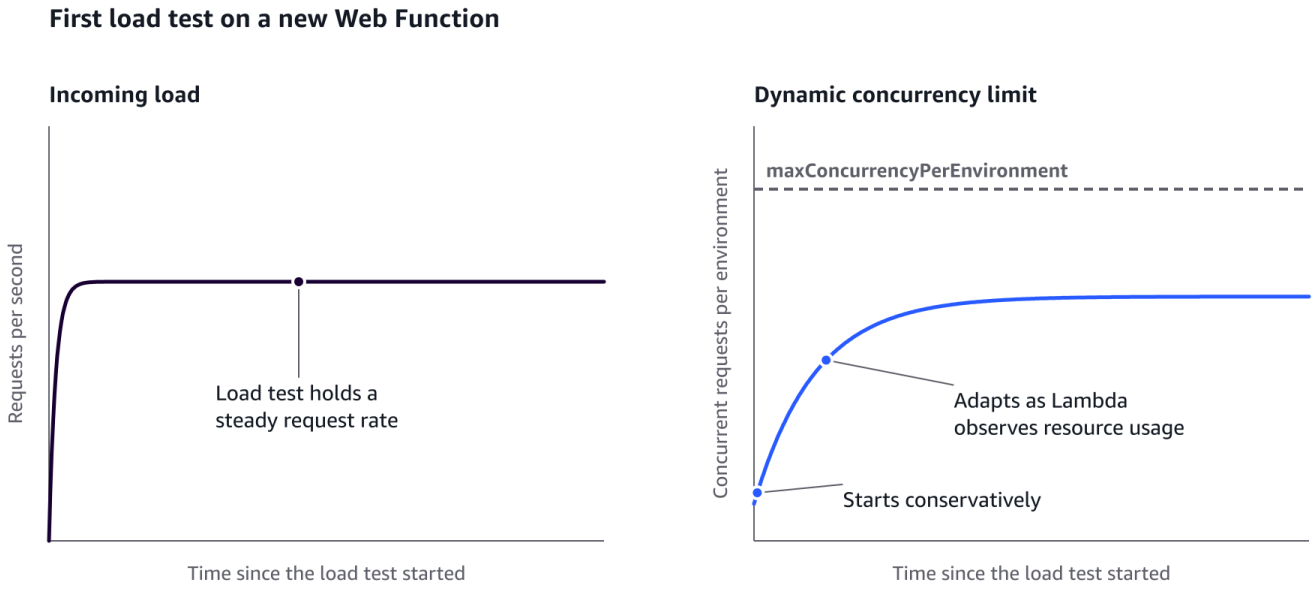


Batas konkurensi dinamis dari Fungsi Web baru

Batas konkurensi dinamis per lingkungan dimulai secara konservatif untuk Fungsi Web baru. Saat fungsi menangani lalu lintas, Lambda belajar dari penggunaan sumber dayanya dan menyesuaikan batasnya. Lambda mengingat apa yang dipelajarinya ketika fungsi dinaikkan ke nol, sehingga lingkungan yang mulai nanti mendapat manfaat dari lalu lintas sebelumnya.

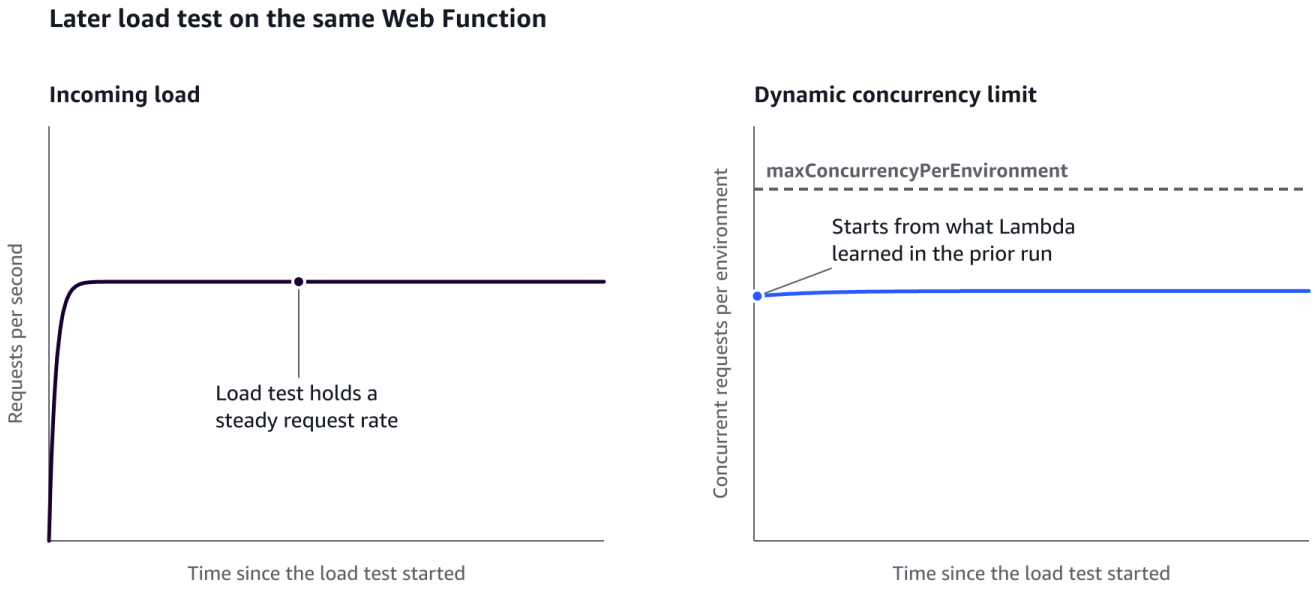
Bagan berikut menunjukkan uji beban pertama pada Fungsi Web baru. Beban masuk naik dengan cepat ke tingkat permintaan yang stabil dan menahannya. Batas konkurensi dinamis dimulai rendah,

meningkat saat Lambda menyesuaikannya berdasarkan penggunaan sumber daya fungsi, dan level turun di bawah `maxConcurrencyPerEnvironment`.



Karena Lambda mengingat apa yang dipelajarinya dari lalu lintas sebelumnya, tes beban selanjutnya pada Fungsi Web yang sama tidak dimulai secara konservatif.

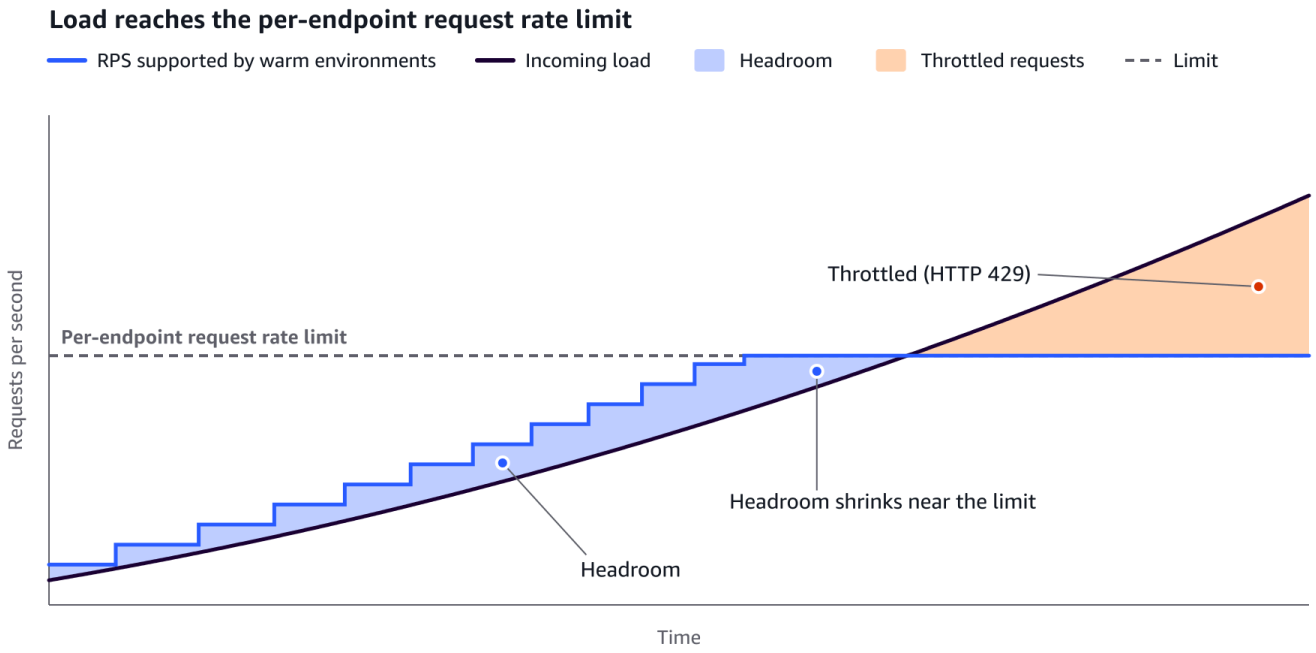
Bagan berikut menunjukkan uji beban selanjutnya pada Fungsi Web yang sama. Beban masuk naik ke tingkat permintaan stabil yang sama, dan batas konkurensi dinamis dimulai pada level yang dipelajari Lambda pada proses sebelumnya.



Karena batas dimulai secara konservatif, uji beban pertama pada Fungsi Web baru mungkin menggunakan lebih banyak lingkungan eksekusi daripada yang dijalankan nanti. Untuk mengukur kinerja yang diharapkan fungsi Anda di bawah lalu lintas yang stabil, jalankan uji beban lebih dari sekali dan gunakan hasil dari proses selanjutnya.

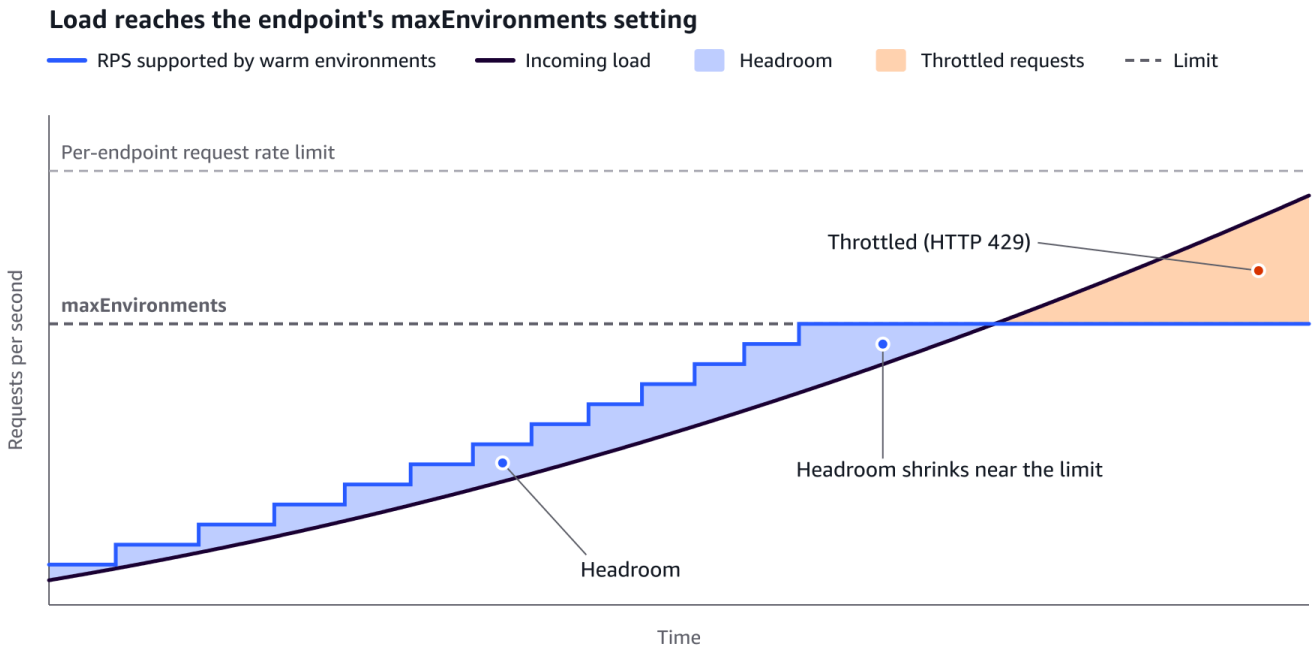
Mencapai batas tingkat permintaan per titik akhir

Batas tingkat permintaan membatasi berapa banyak beban yang dapat diterima titik akhir, sehingga kapasitas tidak tumbuh melewatinya. Saat beban mendekati kuota tarif per titik akhir atau Anda `rateLimit` pengaturan, ruang kepala menyusut. Ketika beban melebihi batas, Lambda mengembalikan HTTP 429 untuk permintaan di atasnya. Untuk menerima lebih banyak permintaan, tingkatkan `rateLimit`, hingga kuota tarif per titik akhir.



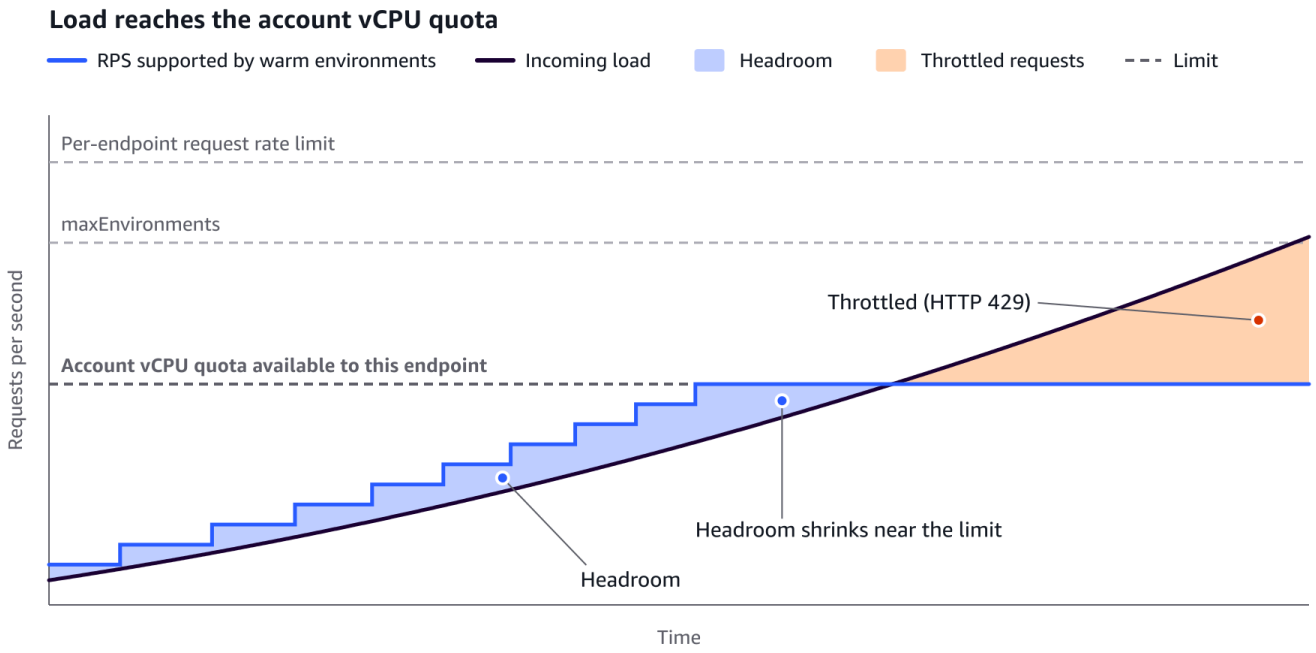
Mencapai MaxEnvironments

Dalam contoh ini, `maxEnvironments` lebih rendah dari batas tingkat permintaan. Ketika titik akhir mencapai `maxEnvironments`, Lambda berhenti menambahkan lingkungan eksekusi. Beban dapat terus tumbuh, sehingga ruang kepala menyusut. Ketika beban melebihi apa yang dapat ditangani lingkungan, Lambda membatasi permintaan baru dengan HTTP 429, meskipun tingkat permintaan masih di bawah batas kecepatan. Untuk memungkinkan lebih banyak kapasitas, tingkatkan `maxEnvironments`.



Mencapai kuota akun vCPU

Semua titik akhir dalam akun berbagi kuota vCPU di masing-masing Wilayah AWS. Lingkungan yang digunakan oleh titik akhir lain mengurangi seberapa jauh titik akhir ini dapat diskalakan. Dalam contoh ini, kuota vCPU yang tersedia untuk titik akhir lebih rendah dari keduanya `maxEnvironments` pengaturan dan batas tingkat permintaannya. Ketika akun mencapai kuota, Lambda berhenti menambahkan lingkungan dan membatasi permintaan yang melebihi kapasitas yang tersedia, meskipun titik akhir belum mencapai batasnya sendiri. Untuk memungkinkan lebih banyak kapasitas, minta peningkatan kuota vCPU.



Siklus hidup lingkungan eksekusi

Lingkungan eksekusi adalah instance komputasi terisolasi yang menjalankan server HTTP Anda. AWS Lambda menciptakan lingkungan eksekusi untuk Fungsi Web Lambda Anda saat lalu lintas tiba. Ini merutekan permintaan ke lingkungan aktif dan menghapus lingkungan yang tidak lagi diperlukan. Memahami siklus hidup ini membantu Anda mengoptimalkan kinerja startup dan menulis kode yang menangani inisialisasi dan shutdown dengan benar.

Fase siklus hidup

Setiap lingkungan eksekusi bergerak melalui tahapan berikut:



Fase	Apa yang terjadi	Permintaan penanganan
Inisialisasi	Lambda memuat kode Anda, menyiapkan runtime, dan memulai proses server HTTP Anda. Server Anda harus mulai mendengar kan alamat `AWS_LAMBDA_HTTP_ENDPOINT` dalam waktu 10 detik.	Lingkungan tidak dapat melayani permintaan selama inisialisasi.
Siap	Server HTTP Anda mendengarkan dan lingkungan tersedia untuk menerima permintaa n. Lambda merutekan permintaan masuk ke lingkungan ini.	Lingkungan menerima permintaan hingga `maxConcurrencyPerE nvironment` batasnya.
Matikan	Lambda berhenti merutekan permintaan baru ke lingkungan, memungkinkan permintaan dalam penerbangan selesai dengan upaya terbaik, mengirim`SIGTERM`, dan kemudian mengakhiri lingkungan.	Tidak ada permintaan baru yang diarahkan ke lingkunga n. In-flight permintaan dapat dilanjutkan, tetapi penyelesa ian tidak dijamin. Setelah itu`SIGTERM`, proses memiliki waktu hingga 1 detik untuk menyelesaikan pembersihan. Keadaan proses hilang.

Inisialisasi (start dingin)

Ketika Lambda menciptakan lingkungan eksekusi baru, fase inisialisasi mencakup langkah-langkah berikut:

1. Lambda memuat kode dari paket penerapan.zip Anda.
2. Lambda menyiapkan lingkungan runtime (termasuk Node.js dan variabel lingkungan Anda).
3. Lambda memulai proses aplikasi Anda.
4. Server HTTP Anda mulai mendengarkan di host dan port yang ditentukan oleh variabel `AWS_LAMBDA_HTTP_ENDPOINT` lingkungan. Itu harus mulai mendengarkan dalam 10 detik.
5. Lambda mendeteksi bahwa server Anda mendengarkan dan menandai lingkungan sebagai siap.

Awal yang dingin terjadi ketika Lambda harus membuat lingkungan baru untuk menangani permintaan karena tidak ada lingkungan yang tersedia. Permintaan pertama setelah cold start mengalami latensi tambahan dari fase inisialisasi. Permintaan selanjutnya ke lingkungan yang sama tidak menimbulkan biaya ini.

Untuk mengurangi dampak awal dingin:

- Minimalkan ukuran paket penerapan Anda. Paket yang lebih kecil memuat lebih cepat.
- Kurangi waktu startup server. Tunda inisialisasi berat (seperti memuat file data besar) setelah server mulai mendengarkan. Lambda merutekan permintaan ke lingkungan segera setelah server mendengarkan, jadi buat penangan permintaan yang bergantung pada pekerjaan yang ditangguhkan menunggu sampai selesai, misalnya dengan menunggu janji bersama.
- Gunakan inisialisasi malas untuk koneksi database dan klien layanan eksternal. Buat koneksi pada permintaan pertama daripada saat startup.

Scale-down dan shutdown

Lambda memantau pemanfaatan lingkungan titik akhir Anda dan menghapus lingkungan yang tidak dibutuhkan, misalnya saat lalu lintas menurun. Sebelum menghapus lingkungan, Lambda menggunakan urutan shutdown berikut:

1. Lambda berhenti merutekan permintaan baru ke lingkungan.
2. Lambda memungkinkan permintaan dalam penerbangan diselesaikan dengan upaya terbaik.
3. Lambda mengirim. `SIGTERM` Proses ini memiliki waktu hingga 1 detik untuk menyelesaikan pembersihan sebelum Lambda menghentikannya dan menghilangkan lingkungan.

Ketika tidak ada lalu lintas, Lambda menskalakan Fungsi Web Anda ke lingkungan nol. Permintaan berikutnya setelah periode skala-ke-nol memicu awal dingin yang baru.

Jika aplikasi Anda perlu melakukan pembersihan selama shutdown, tangani. `SIGTERM` Pembersihan harus selesai dalam 1 detik. Jangan mengandalkan callback shutdown yang dijalankan, karena Lambda tidak dapat menjamin shutdown yang anggun di semua mode kegagalan, misalnya kegagalan perangkat keras yang mendasarinya.

Contoh berikut menangani`SIGTERM`:

```
process.on('SIGTERM', () => {
```

```
  // Finish short cleanup, such as flushing buffered logs, within 1 second.
  process.exit(0);
});
```

Penggunaan kembali lingkungan

Lambda menggunakan kembali lingkungan eksekusi di beberapa permintaan untuk menghindari start dingin yang berulang. Satu lingkungan dapat melayani ribuan permintaan selama masa pakainya. Ini berarti:

- Status global tetap ada - Variabel, cache, dan koneksi yang diinisialisasi server Anda saat startup tetap ada di seluruh permintaan. Anda dapat menggunakan ini untuk caching dalam memori atau penyatuan koneksi.
- Status sistem file tetap ada — File yang ditulis kode Anda ke sistem file lokal (seperti`/tmp`) bertahan hingga Lambda menghapus lingkungan.
- Tidak ada jaminan seumur hidup - Lambda dapat menghapus lingkungan kapan saja. Jangan bergantung pada lingkungan yang berlangsung selama durasi tertentu. Pertahankan data penting ke toko eksternal seperti DynamoDB atau Amazon S3.

Penanganan permintaan bersamaan

Setiap lingkungan eksekusi Web Function menangani beberapa permintaan HTTP bersamaan. `maxConcurrencyPerEnvironment`Pengaturan pada revisi mengontrol jumlah maksimum permintaan simultan per lingkungan.

Karena server Anda menangani beberapa permintaan secara bersamaan, kode Anda harus aman untuk digunakan bersamaan:

- Jangan menyimpan status per permintaan dalam variabel global. Gunakan variabel cakupan permintaan atau konteks middleware sebagai gantinya.
- Jaga status bisa berubah bersama, seperti cache atau penghitung, yang Anda modifikasi melintasi batas. `await` Permintaan interleave pada loop acara. Jika urutan baca-modify-tulis mencakup operasi asinkron, permintaan dalam penerbangan lainnya dapat membaca atau mengubah status sebelum urutan selesai.
- Ketahuilah bahwa Node.js menggunakan loop peristiwa single-threaded. CPU-intensive operasi sinkron memblokir semua permintaan bersamaan di lingkungan itu. Gunakan asinkron I/O dan hindari pemblokiran loop peristiwa.

Untuk informasi selengkapnya tentang pengaturan konkurensi dan pengoptimalan, lihatthe section called “Penskalaan”.

Runtime Fungsi Web Lambda

Lambda Web Functions menjalankan kode server HTTP Anda menggunakan runtime terkelola. Aplikasi Anda harus memulai server HTTP yang mendengarkan pada host dan port yang ditentukan oleh variabel `AWS_LAMBDA_HTTP_ENDPOINT` lingkungan. Fungsi Web Lambda saat ini mendukung. Node.js Secara default, runtime dimuat `index.js` di root paket penerapan. Untuk menggunakan file entri yang berbeda, atur variabel `AWS_LAMBDA_NODEJS_ENTRYPOINT` lingkungan. Lihat perinciannya di the section called “Variabel lingkungan”.

Didukung Node.js versi

Pengidentifikasi runtime	Versi Node.js	Status
`nodejs24.x`	Node.js24	Didukung

Tentukan pengidentifikasi runtime di `runtimeConfig` bagian saat Anda membuat revisi. Contoh:

```
"runtimeConfig": { "runtime": "nodejs24.x" }
```

Runtime `nodejs24.x` Web Function mengikuti siklus penghentian yang sama dengan runtime Lambda Event Functions yang sesuai. Untuk tanggal penghentian dan proses penghentian, lihat. the section called “Kebijakan penghentian waktu proses”

Penambalan runtime otomatis

Lambda secara otomatis menambal Node.js runtime dengan pembaruan keamanan dan perbaikan bug. Patching berlaku untuk layer runtime yang mendasarinya dan tidak mengubah kode aplikasi Anda. `runtime`Pengidentifikasi revisi Anda (misalnya,`nodejs24.x`) selalu menyelesaikan ke versi patch terbaru dari runtime utama tersebut.

> **Note**
>
> Kontrol manajemen runtime tidak didukung untuk Fungsi Web. Anda tidak dapat menyematkan ke versi tambalan tertentu atau menonaktifkan penambalan otomatis. Lambda selalu menggunakan versi patch terbaru dari runtime yang Anda tentukan.

Untuk informasi selengkapnya tentang cara Lambda mengelola pembaruan runtime, lihat. the section called “Pembaruan versi runtime”

Kerangka kerja web yang didukung

Anda dapat menggunakan kerangka Node.js web apa pun yang memulai server HTTP. Kerangka kerja populer meliputi:

- Express
- Next.js
- Fastify
- Koa

Anda tidak memerlukan AWS Lambda Web Adapter atau adaptor event-to-HTTP lainnya. Aplikasi Anda menangani permintaan HTTP secara langsung menggunakan routing standar kerangka kerja dan middleware. Beberapa kerangka kerja, seperti Next.js dalam mode mandiri, membaca variabel lingkungan mereka sendiri untuk alamat mendengarkan. Untuk kerangka kerja ini, gunakan file `index.js` entri yang menetapkan variabel-variabel tersebut `AWS_LAMBDA_HTTP_ENDPOINT` sebelum memulai server. Lihat perinciannya di the section called “Variabel lingkungan”. Untuk aplikasi lengkap, lihatthe section called “Minimal Ekspres application”.

Node.js model eksekusi

Node.jsmenggunakan model eksekusi yang digerakkan oleh peristiwa. Secara default, Node.js menjalankan JavaScript kode Anda pada satu utas utama, dan I/O operasi (panggilan jaringan, akses sistem file) tidak memblokir dan didorong oleh peristiwa. Aplikasi Anda dapat membuat thread tambahan dengan `worker_threads` modul jika membutuhkannya. Model ini memiliki implikasi penting untuk Fungsi Web:

Konkurensi melalui async I/O

Server Anda menangani beberapa permintaan bersamaan dengan menggunakan operasi asinkron (`async`/`await`, Promises, callback). Sementara satu permintaan menunggu respons database, loop peristiwa melayani permintaan lainnya. Inilah sebabnya mengapa Web Functions dapat menangani banyak permintaan bersamaan dalam satu lingkungan tanpa banyak utas.

Hindari memblokir loop acara

CPU-intensive operasi sinkron (penguraian JSON besar, perhitungan kompleks, pembacaan file sinkron) memblokir loop peristiwa dan mencegah semua permintaan lainnya diproses. Jika aplikasi Anda CPU-heavy berfungsi, turunkan ke thread pekerja atau kurangi `maxConcurrencyPerEnvironment` untuk membatasi dampaknya.

Minta isolasi

Karena semua permintaan berbagi proses dan loop peristiwa yang sama, hindari menyimpan status per permintaan dalam variabel global. Gunakan konteks middleware dengan cakupan permintaan (Express`req`seperti/ `res` objek) untuk mengisolasi data permintaan.

Untuk panduan tentang konfigurasi `maxConcurrencyPerEnvironment` berdasarkan karakteristik aplikasi I/O dan CPU Anda, lihatthe section called “Penskalaan”.

Memori dan Node.js tumpukan

Setiap lingkungan eksekusi memiliki memori 2 GB. Berikut ini berlaku untuk tumpukan Node.js V8:

- Tumpukan default adalah 1.024 MB.
- Untuk menggunakan heap yang lebih besar, atur `NODE_OPTIONS` variabel lingkungan revisi, misalnya `NODE_OPTIONS=--max-old-space-size=1536` (nilainya dalam MB).
- Nilai yang Anda tetapkan `NODE_OPTIONS` menggantikan default. Jika Anda hanya menyetel`--max-old-space-size`, Lambda tidak menyetel`--max-semi-space-size`, dan Node.js menggunakan defaultnya sendiri untuk itu.
- Heap, buffer, alokasi asli, runtime, dan permintaan bersamaan semuanya harus sesuai dengan memori lingkungan 2 GB.

Bagaimana Lambda merutekan permintaan ke aplikasi Anda

Titik akhir Fungsi Web hanya menerima lalu lintas HTTPS, pada port 443. HTTP biasa tidak disajikan, jadi Anda tidak perlu mengarahkan HTTP ke HTTPS atau mengonfigurasi TLS di aplikasi Anda.

Lambda mengakhiri TLS di titik akhir dan meneruskan setiap permintaan ke server HTTP aplikasi Anda di dalam lingkungan eksekusi. Aplikasi Anda menyajikan HTTP biasa pada alamat yang disediakan Lambda; Anda tidak pernah menangani sertifikat atau konfigurasi TLS dalam kode Anda.

Lambda memilih host dan port yang harus didengarkan server Anda. Lambda mengkomunikasikan alamat ke aplikasi Anda melalui variabel `AWS_LAMBDA_HTTP_ENDPOINT` lingkungan, dalam format. `host:port` Gunakan variabel ini, yang ditetapkan Lambda, untuk menentukan di mana server Anda mendengarkan. Aplikasi Anda harus membaca variabel saat startup:

```
const [host, port] = (process.env.AWS_LAMBDA_HTTP_ENDPOINT ||
 'localhost:3000').split(':');
app.listen(port, host);
```

Lambda tidak mengatur variabel di komputer lokal Anda. Alamat fallback hanya berlaku ketika variabel tidak disetel, sehingga hanya digunakan dalam proses lokal. Untuk menggunakan alamat lokal yang berbeda, atur `AWS_LAMBDA_HTTP_ENDPOINT` sendiri. Sebagai contoh, lihat the section called “Jalankan server HTTP secara lokal”.

Lambda menambahkan `lambda-web-request-id` header ke setiap permintaan yang diteruskan ke aplikasi Anda, dan mengembalikan nilai yang sama sebagai header respons, sehingga Anda dapat menghubungkan permintaan klien dengan log aplikasi Anda.

Lambda merutekan permintaan web ke alamat yang ditentukan oleh. `AWS_LAMBDA_HTTP_ENDPOINT` Jika server Anda mendengarkan pada host atau port yang berbeda, permintaan tidak mencapai aplikasi Anda.

Server Anda harus mulai mendengarkan dalam waktu 10 detik setelah startup, dan setiap permintaan yang dilayaninya terikat oleh `timeoutSeconds` pengaturan revisi. Untuk urutan startup dan shutdown, lihatthe section called “Siklus hidup lingkungan”. Untuk cara mengkonfigurasi`timeoutSeconds`, lihatthe section called “Revisi”.

Mengkonfigurasi Fungsi Web Lambda

Anda mengonfigurasi sebagian besar pengaturan Fungsi Web pada revisi, yang merupakan snapshot yang tidak dapat diubah dari kode dan konfigurasi aplikasi Anda. Karena revisi tidak dapat diubah,

mengubah pengaturan revisi berarti membuat revisi baru dengan nilai yang diperbarui. Anda menetapkan tag dan kebijakan berbasis sumber daya pada Fungsi Web. Revisi dan titik akhir mewarisi tagnya, dan kebijakan berbasis sumber daya mengontrol siapa yang dapat bertindak pada Fungsi Web, revisinya, dan titik akhirnya. Pengaturan titik akhir berlaku untuk setiap titik akhir, termasuk jenis otentikasi, mode penerapan otomatis, bobot revisi, penskalaan, dan pelambatan. Anda dapat mengubahnya tanpa membuat revisi baru. Topik berikut menjelaskan pengaturan yang dapat Anda konfigurasikan.

Topik

- Paket penyebaran Fungsi Web
- Arsitektur set instruksi
- Batas waktu permintaan
- Variabel lingkungan
- Menandai Fungsi Web
- Menggunakan titik akhir metadata

Paket penyebaran Fungsi Web

Anda mengemas kode Fungsi Web Anda sebagai file.zip dan menyimpannya dalam bucket Amazon S3 berversi. Revisi `codeConfig` mereferensikan bucket, kunci, dan opsional versi objek. Jika Anda menghilangkan versi, Lambda menggunakan versi terbaru pada waktu pembuatan, sehingga setiap revisi menunjuk ke versi tertentu dari paket Anda. Mengunggah paket baru ke kunci yang sama tidak mengubah revisi yang ada, dan Anda dapat memutar kembali ke revisi sebelumnya selama versi objeknya tetap ada di bucket.

Paket harus menyertakan file entri aplikasi Anda dan semua dependensinya (misalnya,`node_modules`). Secara default, Lambda memuat `index.js` di root paket sebagai file entri dan menjalankannya untuk memulai server HTTP Anda. Untuk menggunakan nama atau jalur yang berbeda (misalnya,`dist/index.js`), atur variabel `AWS_LAMBDA_NODEJS_ENTRYPOINT` lingkungan; lihatthe section called “Variabel lingkungan”.

Untuk cara membuat dan mengunggah paket ke Amazon S3 dan membuat revisi yang mereferensikannya, lihat dan. the section called “Kode di Amazon S3” the section called “Buat Fungsi Web pertama Anda”

Arsitektur set instruksi

Fungsi Web berjalan secara eksklusif pada arsitektur Arm64, didukung oleh prosesor AWS Graviton. Tidak seperti Lambda Event Functions, Anda tidak dapat memilih set instruksi: tidak ada pengaturan arsitektur untuk dikonfigurasi.

Setiap lingkungan eksekusi Web Function menggunakan 2 vCPU Arm64. Kuota per wilayah membatasi kapasitas vCPU Arm64 akun Anda. Untuk kuota, lihatthe section called “Fungsi Web Lambda”. Untuk metrik penggunaan yang menunjukkan konsumsi Anda terhadapnya, lihatthe section called “Penggunaan regional dan metrik pelambatan”.

Bangun dependensi asli untuk Linux di arm64 (aarch64). Murni Node.js atau JavaScript kode tidak perlu diubah. Ketergantungan apa pun dengan binari asli harus menyediakan build Linux arm64. Bangun ketergantungan pada mesin Linux arm64 atau dalam wadah Linux arm64, atau kompilasi silang untuk Linux arm64. Untuk contoh yang menginstal dependensi dalam kontainer Arm64, lihat. the section called “Uji paket penerapan”

Batas waktu permintaan

`timeoutSeconds`Pengaturan mengontrol waktu maksimum permintaan HTTP tunggal dapat dijalankan sebelum Lambda menghentikannya. Nilai yang valid adalah 3—900 detik (15 menit). Waktu default-nya adalah 30 detik.

Pilih nilai batas waktu berdasarkan permintaan terlama yang diharapkan dari aplikasi Anda. Menyetel batas waktu terlalu rendah menyebabkan permintaan yang berjalan lama gagal. Mengaturnya terlalu tinggi memungkinkan permintaan macet berjalan lebih lama sebelum Lambda menghentikannya.

Ketika permintaan non-streaming melebihi batas waktu, Lambda mengembalikan HTTP 504 dengan header. `lambda-web-error-type` `Runtime.RequestTimeout` Untuk respons streaming, batas waktu membatasi seluruh aliran. Ketika batas waktu berlalu, Lambda mengakhiri aliran, dan klien menerima kode status yang sudah dikirim, biasanya 200, dengan badan terpotong dan tanpa header. `lambda-web-error-type`

Variabel lingkungan

Variabel lingkungan adalah pasangan nilai kunci yang tersedia untuk kode aplikasi Anda saat runtime. Gunakan variabel lingkungan untuk konfigurasi non-sensitif yang ingin Anda ubah tanpa memodifikasi kode, seperti flag fitur, tingkat log, dan pengidentifikasi sumber daya seperti nama host

database. Untuk nilai sensitif seperti kunci API dan kredenal database, jangan menyimpannya dalam variabel lingkungan. Sebagai gantinya, gunakan Secrets Manager atau AWS Systems Manager Parameter Store dan ambil nilai saat runtime.

Variabel lingkungan diatur saat Anda membuat revisi dan tidak dapat diubah setelah pembuatan. Untuk mengubah variabel lingkungan, buat revisi baru. Dalam revisi baru, tentukan setiap variabel lingkungan dan setiap `serviceConfig` pengaturan yang ingin Anda pertahankan, tidak hanya nilai yang berubah. Pengaturan yang Anda hilangkan mengambil nilai defaultnya, dan variabel lingkungan yang Anda hilangkan tidak disetel.

Lambda menetapkan variabel lingkungan berikut khusus untuk Fungsi Web:

- **`AWS_LAMBDA_HTTP_ENDPOINT`**— Host dan port yang harus didengarkan server HTTP Anda, dalam `host:port` format. Lambda menetapkan variabel ini dan merutekan permintaan ke alamat itu. Baca di kode aplikasi Anda dan ikat persis host dan port itu. Anda tidak dapat mengatur variabel ini dalam variabel lingkungan revisi. Selalu mengikat ke alamat dalam variabel ini, bukan `localhost` atau port hardcode. Runtime menunggu server Anda menerima koneksi di alamat ini, dan inisialisasi gagal jika tidak pernah terjadi. Gunakan alamat fallback hanya untuk proses lokal. Dalam Fungsi Web, Lambda selalu menetapkan variabel ini.

Fungsi Web juga menerima variabel lingkungan runtime Lambda lainnya, sepertiAWS_LAMBDA_FUNCTION_MEMORY_SIZE,, dan. AWS_LAMBDA_METADATA_API AWS_LAMBDA_METADATA_TOKEN Untuk daftar umum, lihatVariabel lingkungan runtime yang ditetapkan. Jika kode atau pustaka membaca variabel dari daftar tersebut, konfirmasikan bahwa Lambda menyetelnya dalam Fungsi Web sebelum Anda mengandalkannya. Fungsi Web tidak menerima setiap variabel dalam daftar itu.

Nama variabel lingkungan berikut dicadangkan untuk Lambda Web Functions dan tidak dapat diatur. Jika Anda menentukan satu, Lambda menolak permintaan untuk membuat Fungsi Web atau revisi:

- `AWS_ACCESS_KEY`, `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`, `AWS_SECRET_KEY`, `AWS_SECURITY_TOKEN`, dan `AWS_SESSION_TOKEN`
- `AWS_DEFAULT_REGION` dan `AWS_REGION`
- `AWS_LAMBDA_HTTP_ENDPOINT`, `AWS_LAMBDA_RUNTIME_API`, `AWS_LAMBDA_INITIALIZATION_TYPE`, dan `AWS_LAMBDA_FUNCTION_MEMORY_SIZE`
- `AWS_LAMBDA_METADATA_API` dan `AWS_LAMBDA_METADATA_TOKEN`

Beberapa kerangka kerja membaca variabel lingkungan mereka sendiri untuk alamat mendengarkan sebagai gantinya. Misalnya, mode Next.js mandiri membaca `HOSTNAME` dan`PORT`. Untuk kerangka kerja seperti itu, gunakan file `index.js` entri yang menetapkan variabel-variabel tersebut `AWS_LAMBDA_HTTP_ENDPOINT` sebelum memulai server.

Secara default, runtime dimuat `index.js` di root paket deployment Anda sebagai file entri aplikasi Anda. Runtime tidak membaca titik masuk dari`package.json`. Jika file entri Anda memiliki nama atau jalur yang berbeda (misalnya,`dist/index.js`), atur variabel `AWS_LAMBDA_NODEJS_ENTRYPOINT` lingkungan ke jalurnya, relatif terhadap root paket Anda.

Contoh berikut menunjukkan cara membaca variabel `AWS_LAMBDA_HTTP_ENDPOINT` lingkungan dalam Node.js aplikasi:

```
// Lambda provides the address to bind in AWS_LAMBDA_HTTP_ENDPOINT ("host:port").
// The fallback applies only when running locally.
const [host, port] = (process.env.AWS_LAMBDA_HTTP_ENDPOINT ||
 'localhost:3000').split(':');
app.listen(port, host, () => {
  console.log(`Server listening on ${host}:${port}`);
});
```

Untuk mengatur variabel lingkungan Anda sendiri, tambahkan `environmentVariables` peta ke `serviceConfig` objek saat Anda membuat revisi, seperti pada contoh berikut. Untuk perintah lengkapnya, lihatthe section called “Membuat revisi”.

```
{
  "executionRoleArn": "arn:aws:iam::111122223333:role/my-web-function-role",
  "environmentVariables": {
    "DATABASE_URL": "my-database-endpoint",
    "LOG_LEVEL": "info",
    "FEATURE_FLAG_NEW_UI": "true"
  }
}
```

Jika Anda menentukan kunci yang dikelola pelanggan, Lambda mengenkripsi variabel lingkungan saat istirahat dengan kunci itu. Lihat informasi yang lebih lengkap di the section called “Enkripsi saat diam”.

Menandai Fungsi Web

Tag adalah label nilai kunci yang Anda lampirkan ke Fungsi Web untuk mengatur sumber daya dan mengontrol akses. Tag berlaku untuk sumber daya Fungsi Web.

Untuk menambahkan tag saat Anda membuat Fungsi Web, gunakan `tags` parameter`CreateWebFunction`. Untuk menambah, menghapus, atau mencantumkan tag nanti, gunakan `TagResource``UntagResource`, dan `ListTags` operasi.

Anda dapat menggunakan tag dengan cara berikut:

- Organisasi dan pencarian — Kelompokkan dan temukan Fungsi Web berdasarkan atribut seperti proyek, lingkungan, atau tim. Anda dapat menemukan Fungsi Web dengan tagnya hanya di Wilayah asalnya. Untuk informasi selengkapnya, lihat the section called “Visibilitas replika”.
- Attribute-based kontrol akses (ABAC) — Tulis AWS Identity and Access Management (IAM) kebijakan yang mengizinkan atau menolak tindakan berdasarkan tag. Gunakan `aws:ResourceTag/${TagKey}` untuk tindakan pada Fungsi Web dan revisi dan titik akhir, yang mewarisi tag Fungsi Web. Pada`CreateWebFunction`, Anda juga dapat menggunakan `aws:RequestTag/${TagKey}` dan`aws:TagKeys`. Anda tidak dapat menandai revisi atau titik akhir secara langsung. Untuk informasi selengkapnya, lihat the section called “Izin” dan the section called “Kontrol akses”.

Tag memiliki persyaratan sebagai berikut:

- Tombol tag adalah 1-128 karakter.
- Nilai tag adalah 0-256 karakter.
- Kunci dan nilai dapat berisi huruf, angka, spasi, dan karakter`_ . : / = + - @`.

Menggunakan titik akhir metadata

Titik akhir metadata Lambda adalah API HTTP lokal dalam lingkungan eksekusi. Ini mengembalikan metadata dalam format JSON sederhana, termasuk ID Availability Zone (AZ) tempat lingkungan berjalan. Gunakan untuk merutekan ke sumber daya AZ yang sama seperti Amazon dan ElastiCache Amazon RDS untuk latensi yang lebih rendah. Anda juga dapat menggunakannya untuk menerapkan AZ-aware pola ketahanan.

> **Note**
>
> Fungsi Web menjalankan server web persisten daripada handler per-pemanggilan. Baca titik akhir metadata sekali saat startup aplikasi dan cache nilainya, atau baca per permintaan jika Anda membutuhkannya dalam konteks permintaan. Jangan berasumsi penangan per doa baru. Untuk Fungsi Web Multi-wilayah, titik akhir melaporkan AZ dalam Wilayah AWS yang melayani permintaan.

Bagian-bagian

- Memulai

Memulai

Powertools untuk AWS Lambda menyediakan utilitas untuk mengakses titik akhir metadata Lambda. Utilitas menyimpan respons setelah panggilan pertama. Untuk informasi selengkapnya, lihat Metadata dalam dokumentasi Powertools for AWS Lambda (). TypeScript

Karena Fungsi Web berjalanNode.js, contoh berikut menggunakan utilitas metadata Powertools untuk AWS Lambda untuk membaca ID AZ sekali saat server startup dan cache selama proses berlangsung. Jalankan perintah npm berikut untuk menginstal paket Powertools dan Express:

```
npm install @aws-lambda-powertools/commons express
```

Contoh berikut digunakan Express untuk ilustrasi. Pendekatan yang sama bekerja dengan kerangka Node.js web apa pun; lihatthe section called “Kerangka kerja web yang didukung”. Baca ID AZ saat startup dan gunakan kembali nilai cache saat menangani permintaan:

Example Mengambil ID AZ dengan Powertools saat startup () TypeScript

```
import express from 'express';
import { getMetadata } from '@aws-lambda-powertools/commons/utils/metadata';

const app = express();

// Read the AZ ID once at startup and cache it for the life of the server.
const { AvailabilityZoneID: azId } = await getMetadata();

// Lambda provides the address to bind in AWS_LAMBDA_HTTP_ENDPOINT ("host:port").
```

```
// The fallback applies only when running locally.
const [host, port] = (process.env.AWS_LAMBDA_HTTP_ENDPOINT ||
 'localhost:3000').split(':');

app.get('/', (req, res) => {
  res.json({ azId });
});

app.listen(port, host, () => {
  console.log(`Server listening on ${host}:${port}, AZ: ${azId}`);
});
```

Anda juga dapat memanggil titik akhir metadata secara langsung. Lambda secara otomatis menetapkan variabel `AWS_LAMBDA_METADATA_API` dan `AWS_LAMBDA_METADATA_TOKEN` lingkungan di setiap lingkungan eksekusi, dan Anda menyertakan token dengan skema Bearer di header. `Authorization` Untuk URL titik akhir, format permintaan dan respons, serta respons kesalahan, lihatthe section called “Titik akhir metadata”.

Misalnya, baca alamat server metadata dari `AWS_LAMBDA_METADATA_API` dan kirim permintaan GET yang diautentikasi:

```
METADATA_ENDPOINT="http://${AWS_LAMBDA_METADATA_API}/2026-01-15/metadata/execution-
environment"
RESPONSE=$(curl -s -H "Authorization: Bearer ${AWS_LAMBDA_METADATA_TOKEN}"
 "$METADATA_ENDPOINT")
AZ_ID=$(echo "$RESPONSE" | jq -r '.AvailabilityZoneID')
```

Titik akhir kembali`application/json`. Misalnya: `{"AvailabilityZoneID": "use1-az1"}`.

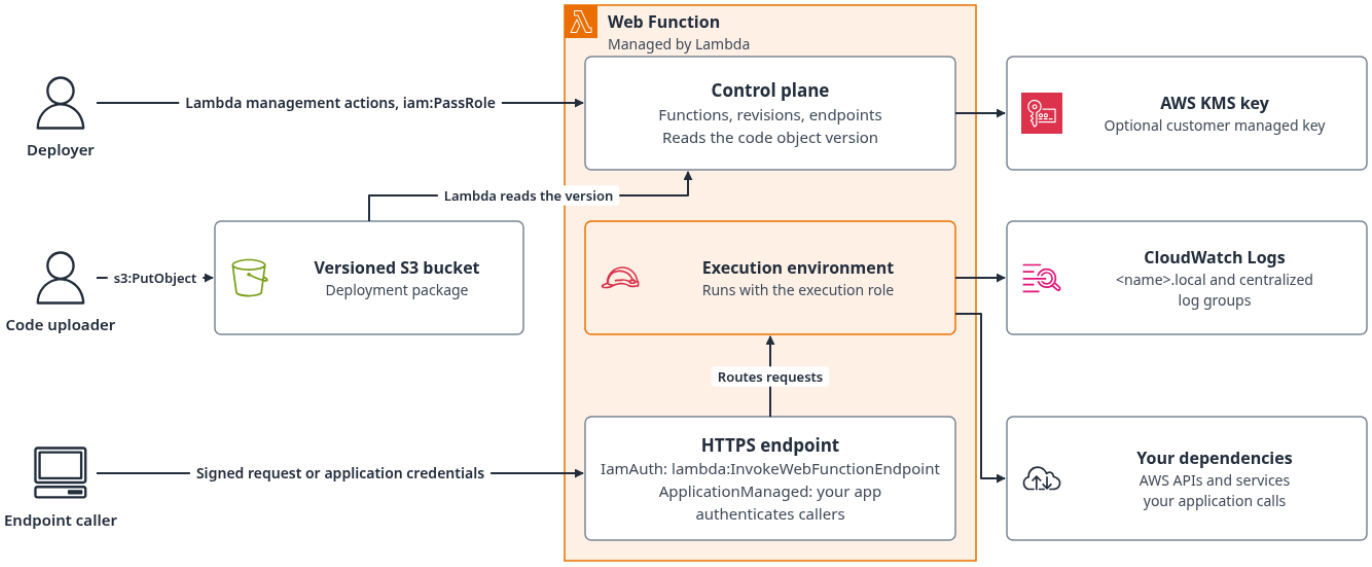
Izin Fungsi Web Lambda

Lambda Web Functions use AWS Identity and Access Management (IAM) untuk mengontrol apa yang dapat dilakukan kode aplikasi Anda (peran eksekusi) dan siapa yang dapat mengakses Fungsi Web Anda (kebijakan berbasis sumber daya). Halaman ini menjelaskan cara mengkonfigurasi setiap jenis.

- Peran eksekusi - Peran IAM yang diasumsikan kode Fungsi Web Anda untuk mengakses AWS layanan seperti Amazon CloudWatch Logs, DynamoDB, atau Amazon S3.
- Resource-based kebijakan — Kebijakan yang dilampirkan pada Fungsi Web Anda. Ini mendefinisikan prinsip atau AWS layanan IAM mana yang dapat melakukan tindakan Fungsi

Web, seperti memanggil titik akhir yang jenis otentikasi, yang berarti Lambda mengautentikasi permintaan dengan IAM. `IamAuth` Untuk informasi selengkapnya, lihat the section called “Jenis autentikasi”. Prinsipal IAM di akun lain juga membutuhkan izin berbasis identitas.

Diagram berikut menunjukkan identitas yang memerlukan izin dan apa yang digunakan masing-masing. Deployer memanggil tindakan dan `iam:PassRole` kebutuhan manajemen Lambda pada peran eksekusi. Pengunggah kode perlu `s3:PutObject` pada bucket Amazon S3 berversi, dan Lambda membaca versi objek kode. Lambda dapat mengenkripsi kode Fungsi Web dan variabel lingkungan Anda dengan kunci terkelola pelanggan opsional. AWS KMS Kode Anda berjalan dengan peran eksekusi, yang memberikan akses ke Amazon CloudWatch Logs dan dependensi Anda. Pemanggil endpoint membutuhkan endpoint, atau kredensi aplikasi `lambda:InvokeWebFunctionEndpoint` untuk `IamAuth` titik akhir. `ApplicationManaged`



Grant each actor only what its role needs. With IamAuth, endpoint access can also come from a resource-based policy.

Peran eksekusi

Peran eksekusi adalah peran IAM yang diasumsikan Lambda saat menjalankan kode Fungsi Web Anda. Aplikasi Anda menggunakan izin peran ini untuk berinteraksi dengan AWS layanan lain. Minimal, Fungsi Web Anda memerlukan izin untuk menulis log ke Amazon CloudWatch Logs sehingga Anda dapat memantau aplikasi Anda. Jika aplikasi Anda membaca dari DynamoDB, menulis ke Amazon S3, atau memanggil layanan AWS lain, Anda menambahkan izin tersebut ke peran eksekusi.

Membuat peran eksekusi

Untuk membuat peran eksekusi untuk Fungsi Web Anda, buat peran IAM dengan kebijakan kepercayaan yang memungkinkan kepala layanan Lambda untuk menerimanya, dan lampirkan kebijakan izin yang memberikan akses yang dibutuhkan aplikasi Anda.

Untuk membuat peran eksekusi (AWS CLI)

1. Buat file bernama `trust-policy.json` dengan kebijakan kepercayaan berikut. Kebijakan ini memungkinkan kepala layanan Lambda untuk mengambil peran atas nama Anda.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "lambda.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

2. Buat peran.

```
aws iam create-role \
    --role-name my-web-function-role \
    --assume-role-policy-document file://trust-policy.json
```

3. Lampirkan kebijakan terkelola peran eksekusi AWS Lambda dasar. Kebijakan ini memberikan izin untuk membuat grup log, aliran log, dan menulis peristiwa log ke Amazon Logs. CloudWatch

```
aws iam attach-role-policy \
    --role-name my-web-function-role \
    --policy-arn arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole
```

Jika Anda lebih suka membuat kebijakan izin kustom daripada menggunakan kebijakan terkelola, izin minimum yang dibutuhkan Fungsi Web untuk pencatatan adalah:

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:us-east-1:111122223333:log-group:/aws/lambda/web/my-web-
app:*",
                "arn:aws:logs:*:111122223333:log-group:/aws/lambda/web/my-web-
app.local:*"
            ]
        }
    ]
}
```

Kebijakan ini mencakup dua grup log. Lambda menulis log In-region ke grup `.local` log dan kemudian memusatkannya ke dalam grup log tanpa akhiran di Wilayah beranda. Grup log terpusat ARN menggunakan Wilayah asal. `.local`ARN menggunakan wildcard Wilayah karena Lambda menulis ke grup itu di setiap Wilayah di mana titik akhir melayani lalu lintas. Ini memungkinkan satu peran eksekusi mendukung titik akhir dengan set Wilayah yang berbeda. Jika peran tidak dapat menulis ke grup `.local` log di Wilayah, Lambda tidak mengirimkan log dari Wilayah tersebut. Jika Anda menggunakan grup log kustom, ganti kedua ARN dengan nama grup log Anda dan nama grup log Anda diikuti oleh`.local`. Untuk informasi selengkapnya, lihat the section called “Grup log”.

Menambahkan izin untuk layanan lain

Jika aplikasi Anda mengakses AWS layanan lain, tambahkan izin yang sesuai ke peran eksekusi. Contoh berikut menambahkan akses baca ke tabel DynamoDB dan akses tulis ke bucket Amazon S3.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
```

```
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:us-east-1:111122223333:log-group:/aws/lambda/web/my-web-
app:*",
                "arn:aws:logs:*:111122223333:log-group:/aws/lambda/web/my-web-
app.local:*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem",
                "dynamodb:Query"
            ],
            "Resource": "arn:aws:dynamodb:us-east-1:111122223333:table/my-table"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject"
            ],
            "Resource": "arn:aws:s3:::my-output-bucket/*"
        }
    ]
}
```

Identitas IAM yang membuat Fungsi Web atau revisi menggunakan izinnya sendiri, bukan peran eksekusi. Identitas itu membutuhkan `lambda:CreateWebFunction` atau`lambda:CreateWebFunctionRevision`, `iam:PassRole` pada peran eksekusi, dan baca akses ke paket penerapan di Amazon S3. Untuk izin Amazon S3, lihat. the section called “Izin penelepon” Jika Anda menggunakan kunci yang dikelola pelanggan untuk mengenkripsi kode Fungsi Web dan variabel lingkungan, identitas itu juga memerlukan AWS KMS izin. Untuk informasi selengkapnya, lihat the section called “Enkripsi saat diam”.

Resource-based kebijakan

Kebijakan berbasis sumber daya adalah dokumen kebijakan JSON yang Anda lampirkan ke Fungsi Web Anda. Ini dapat memberikan tindakan Fungsi Web apa pun kecuali `lambda:CreateWebFunction` untuk kepala sekolah atau layanan IAM. AWS Misalnya, dapat memberikan `lambda:InvokeWebFunctionEndpoint`

`IamAuth` titik akhir. Hal ini juga dapat memberikan tindakan manajemen seperti`lambda:GetWebFunction`,`lambda:CreateWebFunctionRevision`, dan`lambda:UpdateWebFunctionEndpoint`.

Lampiran kebijakan dan pernyataannya menggunakan ARN sumber daya yang berbeda:

- Fungsi Web ARN — Gunakan ARN ini dengan`PutResourcePolicy`,`GetResourcePolicy`, dan`DeleteResourcePolicy`, dan dalam `Resource` elemen pernyataan yang memberikan tindakan pada Fungsi Web itu sendiri, seperti,, atau. `lambda:GetWebFunction` `lambda:CreateWebFunctionRevision` `lambda:CreateWebFunctionEndpoint` Misalnya, `arn:aws:lambda:us-east-1:111122223333:web-function/my-function`. Untuk tindakan pada titik akhir yang ada, seperti`lambda:UpdateWebFunctionEndpoint`, gunakan ARN endpoint. Untuk tindakan pada revisi yang ada, seperti`lambda:DeleteWebFunctionRevision`, gunakan ARN revisi. Untuk mencakup semua titik akhir Fungsi Web, gunakan`arn:aws:lambda:us-east-1:111122223333:web-function/my-function/endpoint/*`.
- Endpoint ARN — Gunakan ARN ini dalam elemen pernyataan `Resource` kebijakan yang memberikan. `lambda:InvokeWebFunctionEndpoint` Misalnya, `arn:aws:lambda:us-east-1:111122223333:web-function/my-function/endpoint/prod`.

Untuk prinsipal IAM di akun yang sama, kebijakan berbasis identitas kepala sekolah atau kebijakan berbasis sumber daya Fungsi Web dapat memberikan izin. Untuk prinsipal IAM di akun lain, kedua kebijakan harus memberikan izin. Ini berlaku untuk tindakan pemanggilan dan manajemen.

Untuk mengizinkan atau menolak tindakan berdasarkan tag Fungsi Web, gunakan kontrol akses berbasis atribut (ABAC). Untuk kunci kondisi yang didukung, lihatthe section called “Tanda”.

> **Note**
>
> Fungsi Web tidak mendukung`AddPermission`,`RemovePermission`, atau`GetPolicy`. Mengelola kebijakan dengan `PutResourcePolicy``GetResourcePolicy`,, dan `DeleteResourcePolicy`

Gunakan kebijakan berbasis sumber daya saat:

- Anda ingin memberikan akses pemanggilan ke prinsipal di Akun AWS (pemanggilan lintas akun) yang berbeda.

- Anda ingin memberikan akses panggilan ke AWS layanan seperti CloudFront.
- Anda ingin membiarkan prinsipal di lain Akun AWS mengelola Fungsi Web, misalnya untuk membuat revisi atau memperbarui titik akhir.
- Untuk pemanggilan akun yang sama, Anda tidak dapat mengubah kebijakan IAM pemanggil (misalnya, saat pemanggil dikelola oleh tim lain).

Kebijakan publik tidak diperbolehkan

Anda tidak dapat melampirkan kebijakan berbasis sumber daya publik ke Fungsi Web. Jika kebijakan bersifat publik, permintaan gagal dengan a`ValidationException`.

Di bagian ini, publik menjelaskan kebijakan yang memberikan akses ke seperangkat prinsip yang tidak ditentukan. Ini tidak menggambarkan aksesibilitas jaringan titik akhir atau jenis otentikasi.

Apa yang membuat kebijakan publik

Kebijakan bersifat publik kecuali membatasi siapa yang dapat memanggil sumber daya ke seperangkat prinsip yang ditentukan dengan baik. Lambda berasumsi bahwa kebijakan tersebut bersifat publik dan kemudian mengevaluasi apakah itu memenuhi syarat sebagai non-publik. Kebijakan bersifat publik jika memungkinkan prinsipal yang dapat diasumsikan oleh setiap AWS pelanggan. Kasus yang paling umum adalah prinsipal wildcard yang tidak dibatasi, seperti pada contoh berikut. Lambda menolak kebijakan ini dengan a. `ValidationException` Jangan gunakan itu.

```
{
  "Effect": "Allow",
  "Principal": "*",
  "Action": "lambda:InvokeWebFunctionEndpoint",
  "Resource": "arn:aws:lambda:us-west-2:123456789012:web-function/my-web-function/
endpoint/prod"
}
```

Pembatasan berikut tidak membuat kebijakan non-publik:

- Kondisi alamat IP. Mempersempit `aws:SourceIp` ke kisaran CIDR tidak cukup.
- Kepala sekolah federasi. `Principal`Blok yang memberi nama penyedia identitas federasi tidak membatasi akses ke kumpulan identitas yang ditentukan dengan baik.

- Kunci kondisi yang menggambarkan sumber daya. Kunci seperti jenis otentikasi titik akhir menjelaskan bagaimana sumber daya dikonfigurasi, bukan siapa yang memanggil, sehingga mereka tidak dapat membatasi prinsipal.

Ada dua cara untuk membuat kebijakan non-publik.

Tentukan prinsipal di blok Principal

Buat daftar satu atau lebih prinsip yang ditentukan sepenuhnya di blok. `Principal` Nilai tidak boleh berisi wildcard atau variabel kebijakan. Jumlah kepala sekolah tidak masalah.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": [
      "123456789012",
      "111122223333"
    ]
  },
  "Action": "lambda:InvokeWebFunctionEndpoint",
  "Resource": "arn:aws:lambda:us-west-2:123456789012:web-function/my-web-function/
endpoint/prod"
}
```

Prinsipal IAM di akun lain juga memerlukan izin berbasis identitas untuk memanggil titik akhir.

Gunakan kunci kondisi tepercaya

Membatasi salah satu dari kunci kondisi berikut untuk nilai tetap. Kunci ini mengidentifikasi pemanggil yang diautentikasi dan tidak dapat diatur oleh pemohon.

- `aws:PrincipalAccount`
- `aws:PrincipalArn`
- `aws:PrincipalOrgID`
- `aws:PrincipalOrgPaths`
- `aws:SourceAccount`
- `aws:SourceArn`
- `aws:SourceOrgID`

- `aws:SourceOrgPaths`
- `aws:SourceVpc`
- `aws:SourceVpcArn`
- `aws:SourceVpce`
- `aws:VpceAccount`
- `aws:VpceOrgID`
- `aws:VpceOrgPaths`
- `aws:Ec2InstanceSourceVpc`
- `aws:UserId`
- `aws:PrincipalServiceName`
- `aws:PrincipalServiceNamesList`
- `aws:PrincipalIsAWSService`
- `lambda:SourceFunctionArn`

Memberi nama kunci tepercaya tidak cukup dengan sendirinya. Anda juga harus membatasi nilainya:

- Nilai string tidak boleh berisi wildcard atau variabel kebijakan.
- Nilai ARN tidak boleh berisi wildcard atau variabel kebijakan di lima bidang pertama ARN:`arn`, partisi, layanan, Wilayah, dan ID akun. Wildcard di bagian sumber daya diperbolehkan.
- `aws:PrincipalIsAWSService` harus diatur ke `true`.

Kebijakan berikut memberi prinsip di organisasi Anda izin berbasis sumber daya yang diperlukan untuk memanggil titik akhir dan tidak dianggap publik. Prinsipal IAM di akun lain juga membutuhkan izin berbasis identitas.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowOrgAccess",
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
```

```
      "Action": "lambda:InvokeWebFunctionEndpoint",
      "Resource": "arn:aws:lambda:us-west-2:123456789012:web-function/my-web-function/
endpoint/prod",
      "Condition": {
        "StringEquals": {
          "aws:PrincipalOrgID": "o-a1b2c3d4e5"
        }
      }
    }
  ]
}
```

Cross-account akses

Kebijakan berbasis sumber daya berikut memberikan peran IAM tertentu dalam izin akun lain untuk memanggil satu titik akhir Fungsi Web Anda. Peran tersebut juga membutuhkan izin berbasis identitas untuk memanggil titik akhir.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowCrossAccountInvoke",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::444455556666:role/caller-role"
            },
            "Action": "lambda:InvokeWebFunctionEndpoint",
            "Resource": "arn:aws:lambda:us-east-1:111122223333:web-function/my-
function/endpoint/prod"
        }
    ]
}
```

Agar pemanggilan lintas akun berhasil, semua hal berikut harus benar:

- Kebijakan berbasis identitas penelepon memberikan izin `lambda:InvokeWebFunctionEndpoint` pada ARN titik akhir.
- Kebijakan berbasis sumber daya pada Fungsi Web memberikan izin pemanggil (seperti yang ditunjukkan di atas).
- Endpoint menggunakan `IamAuth` otentikasi sehingga Lambda mengevaluasi kredensi pemanggil.

Memberikan akses ke AWS layanan

Kebijakan berbasis sumber daya berikut memberikan izin utama CloudFront layanan untuk menjalankan Fungsi Web Anda. Gunakan kebijakan ini saat Anda mengonfigurasi CloudFront distribusi dengan titik akhir Fungsi Web sebagai asal.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowCloudFrontInvoke",
            "Effect": "Allow",
            "Principal": {
                "Service": "cloudfront.amazonaws.com"
            },
            "Action": "lambda:InvokeWebFunctionEndpoint",
            "Resource": "arn:aws:lambda:us-east-1:111122223333:web-function/my-
function/endpoint/prod",
            "Condition": {
                "ArnLike": {
                    "aws:SourceArn":
 "arn:aws:cloudfront::111122223333:distribution/EDFDVBD6EXAMPLE"
                }
            }
        }
    ]
}
```

`aws:SourceArn`Kondisi ini membatasi akses ke CloudFront distribusi tertentu. Tanpa kondisi ini, CloudFront distribusi apa pun di akun apa pun dapat memanggil Fungsi Web Anda.

Replikasi dan konsistensi akhirnya

Saat Anda melampirkan atau memperbarui kebijakan berbasis sumber daya pada Fungsi Web yang memiliki atau MultiRegiontitik akhir PerRegion, Lambda mereplikasi kebijakan tersebut ke semua replika tempat titik akhir beroperasi. Wilayah AWS Replikasi ini pada akhirnya konsisten.

Selama jendela replikasi, Lambda mungkin mengevaluasi permintaan di Wilayah replika terhadap versi kebijakan sebelumnya. Ini berarti:

- Izin yang baru ditambahkan mungkin tidak berlaku di semua Wilayah dengan segera.

- Lambda mungkin masih menghormati izin yang baru saja dihapus di beberapa Wilayah sampai replikasi selesai.

Tidak ada sinyal penyelesaian yang terlihat pelanggan untuk replikasi kebijakan. Uji perilaku izinkan atau tolak yang diharapkan di setiap Wilayah titik akhir sebelum mengandalkan kebijakan baru. Untuk penahanan darurat, perbarui perutean lalu lintas, atau setel titik akhir `throttleConfig.rateLimit` `0` untuk memblokir semua lalu lintas saat perubahan kebijakan menyebar. Untuk informasi selengkapnya, lihat the section called “Throttling”.

Untuk HomeRegiontitik akhir, kebijakan hanya berlaku di Wilayah asal dan tidak memerlukan replikasi Lintas wilayah. Perubahan masih bisa memakan waktu singkat untuk diterapkan.

Izin untuk mengkonfigurasi jaringan

Identitas yang mengonfigurasi jaringan memerlukan izin untuk membuat atau memperbarui sumber daya Lambda yang relevan. Ini juga membutuhkan izin untuk melewati peran dan konektor yang diperlukan. Bergantung pada operasinya, izin meliputi:

- `lambda:CreateNetworkConnector`
- `lambda:UpdateNetworkConnector`
- `lambda:GetNetworkConnector`
- `lambda:CreateWebFunction`
- `lambda:CreateWebFunctionRevision`
- `lambda:UpdateWebFunctionEndpoint`
- `lambda:PassNetworkConnector`
- `iam:PassRole`

Penelepon harus memiliki `lambda:PassNetworkConnector` konektor yang direferensikan setiap kali `create-web-function-revision` panggilan `create-web-function` atau ditetapkan. `egressNetworkConnectorArn` Cakupan izin ini ke sumber daya konektor yang disetujui. Cakupan `iam:PassRole` ke peran eksekusi yang disetujui, yang Anda lewati saat membuat Fungsi Web atau revisi, dan peran operator yang disetujui, yang Anda lewati saat membuat atau memperbarui konektor jaringan. Untuk kebijakan izin peran operator, lihatthe section called “Izin peran operator”.

Kontrol akses titik akhir Fungsi Web Lambda

Anda dapat mengontrol akses ke titik akhir Fungsi AWS Lambda Web Anda menggunakan yang `AuthType` Anda tetapkan pada setiap titik akhir. Bergantung pada jenis autentikasi, Anda juga dapat melampirkan kebijakan berbasis sumber daya ke Fungsi Web untuk mengontrol siapa yang dapat memanggil titik akhir.

Saat Anda mengonfigurasi titik akhir, Anda menentukan salah satu `AuthType` opsi berikut:

`IamAuth`

Dengan jenis otentikasi ini, Lambda AWS Identity and Access Management menggunakan (IAM) untuk mengautentikasi dan mengotorisasi permintaan berdasarkan kebijakan identitas pemanggil dan kebijakan berbasis sumber daya Fungsi Web. Baik pemanggil menandatangani permintaan secara langsung dengan AWS Signature Version 4 (SigV4), atau dengan Sigv4a Sigv4a (Sigv4a) untuk `MultiRegion` titik akhir, atau tanda tangan CloudFront Amazon atas nama mereka saat berada di depan titik akhir (lihat). Layanan AWS the section called “CloudFront integrasi” Pilih opsi ini ketika Anda ingin Lambda menerapkan kontrol IAM-based akses pada titik akhir Anda.

`ApplicationManaged`

Lambda tidak mengautentikasi atau mengotorisasi permintaan. Setiap permintaan mencapai fungsi Anda, dan kode fungsi Anda bertanggung jawab atas otentikasi apa pun. Pilih opsi ini untuk titik akhir publik, atau saat Anda menangani otentikasi dalam kode Anda sendiri.

Tindakan IAM untuk memanggil titik akhir adalah. `lambda:InvokeWebFunctionEndpoint`

Anda menetapkan jenis otentikasi endpoint dengan `--auth-type` parameter saat Anda membuat endpoint (`create-web-function-endpoint`), dan Anda dapat mengubahnya nanti dengan. `update-web-function-endpoint` Untuk perintah create, lihatthe section called “Membuat titik akhir”. Untuk properti yang dapat Anda ubah, lihatthe section called “Memperbarui titik akhir”.

Menggunakan jenis **`IamAuth`**otentikasi

Jika Anda memilih jenis `IamAuth` otentikasi, penelepon harus menandatangani setiap permintaan dengan SigV4 (Sigv4a untuk titik akhir) dan harus diizinkan untuk memanggil `MultiRegion` titik akhir. Bergantung pada siapa yang membuat permintaan, Anda memberikan izin ini dengan kebijakan berbasis sumber daya pada Fungsi Web, kebijakan berbasis identitas pada pemanggil, atau keduanya.

Jika pemanggil Akun AWS sama dengan Fungsi Web, mereka dapat diberikan `lambda:InvokeWebFunctionEndpoint` melalui kebijakan berbasis identitas atau kebijakan berbasis sumber daya fungsi. Jika penelepon berada di akun yang berbeda, mereka memerlukan kebijakan berbasis identitas yang memberikan tindakan dan kebijakan berbasis sumber daya pada fungsi yang memungkinkan mereka.

Resource-based kebijakan dilampirkan ke Fungsi Web, tetapi setiap pernyataan dapat mencakup `Resource` ke titik akhir ARN tertentu. Kebijakan berbasis sumber daya berikut memungkinkan `InvokeRole` peran dalam akun `222233334444` untuk memanggil titik akhir: `prod` `my-function`

Example Cross-account kebijakan berbasis sumber daya

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::222233334444:role/InvokeRole"
            },
            "Action": "lambda:InvokeWebFunctionEndpoint",
            "Resource": "arn:aws:lambda:us-east-1:111122223333:web-function/my-
function/endpoint/prod"
        }
    ]
}
```

Untuk melampirkan kebijakan berbasis sumber daya, gunakan `put-resource-policy` dengan fungsi ARN (bukan ARN titik akhir):

```
aws lambda-web put-resource-policy \
    --resource-arn "arn:aws:lambda:us-east-1:111122223333:web-function/my-function" \
    --policy file://policy.json \
    --region us-east-1
```

Anda dapat meninjau atau menghapus kebijakan dengan `get-resource-policy` dan`delete-resource-policy`.

Fungsi Web memiliki satu kebijakan berbasis sumber daya, dan menggantikannya. `put-resource-policy` Untuk menambahkan pernyataan, seperti CloudFront pernyataan dithe section called

"CloudFront integrasi", dapatkan kebijakan saat iniget-resource-policy, tambahkan pernyataan ke Statement array, dan teruskan revisionId dengan yang dikembalikan--revision-id. Jika seseorang mengubah kebijakan setelah Anda membacanya, permintaan gagal dengan a ConflictException alih-alih menimpa perubahan mereka.

Untuk mengizinkan pemanggilan semua titik akhir fungsi, gunakan wildcard di ARN sumber daya:

```
"Resource": "arn:aws:lambda:us-east-1:111122223333:web-function/my-function/endpoint/*"
```

Ganti Region, ID akun, dan nama fungsi dengan nilai Anda sendiri. Tetap endpoint/* di akhir.

Memberikan izin pemanggilan kepada kepala sekolah

Penelepon lambda:InvokeWebFunctionEndpoint perlu memanggil titik akhir. Untuk pemanggil di akun yang sama dengan Fungsi Web, pemberian tindakan ini dalam kebijakan berbasis identitas pemanggil sudah cukup. Tidak diperlukan kebijakan berbasis sumber daya. Untuk pemanggil di akun lain, kebijakan berbasis identitas ini diperlukan selain kebijakan berbasis sumber daya pada fungsi yang memungkinkan mereka.

Lampirkan kebijakan berikut ke peran IAM atau pengguna yang akan memanggil titik akhir:

Example Identity-based kebijakan untuk memanggil titik akhir

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "lambda:InvokeWebFunctionEndpoint",
            "Resource": "arn:aws:lambda:us-east-1:111122223333:web-function/my-
function/endpoint/prod"
        }
    ]
}
```

Menandatangani permintaan

Lambda tidak menyediakan SDK khusus untuk memanggil titik akhir. Anda mengirim permintaan HTTP standar dan menandatanganinya sendiri dengan SigV4 atau Sigv4a, menggunakan nama layanan. lambda Perhatikan hal-hal berikut:

- Setiap permintaan yang ditandatangani harus menyertakan `X-Amz-Content-SHA256` header, termasuk permintaan tanpa isi. Lambda tidak membaca isi permintaan saat memvalidasi tanda tangan, jadi Lambda menggunakan nilai header ini sebagai hash payload. Permintaan tanpa itu ditolak dengan`403 Forbidden`.
- Jika Anda tidak menandatangani isi (misalnya, saat streaming), gunakan nilai literal `UNSIGNED-PAYLOAD` untuk`X-Amz-Content-SHA256`. Lambda tidak mendukung penandatanganan chunked body.
- Lambda tidak memeriksa isi permintaan terhadap nilainya. `X-Amz-Content-SHA256` Tanda tangan mencakup nilai header, bukan isi isi. Jika Anda membutuhkan perlindungan integritas tubuh di luar TLS, terapkan dalam aplikasi Anda.
- Titik akhir regional (`HomeRegion`,`PerRegion`) menggunakan SigV4 dengan Region titik akhir. `MultiRegion`titik akhir menggunakan AWS Signature Version 4a (Sigv4a), yang menandatangani satu set Wilayah, bukan satu Wilayah. Gunakan set Wilayah`*`.
- Jika `IamAuth` permintaan muncul`403 Forbidden`, periksa apakah permintaan ditandatangani dengan SigV4 (atau Sigv4a), menyertakan `X-Amz-Content-SHA256` header, dan menggunakan kredenal yang belum kedaluwarsa, dan kebijakan Anda mengizinkan ARN titik akhir. `lambda:InvokeWebFunctionEndpoint`

Untuk informasi selengkapnya tentang permintaan penandatanganan, lihat Menandatangani permintaan AWS API.

Setelah Lambda memvalidasi tanda tangan Sigv4 atau Sigv4a, Lambda akan menghapus `Authorization` header sebelum meneruskan permintaan ke fungsi Anda. Jika Anda ingin fungsi Anda menerima `Authorization` header pada `IamAuth` titik akhir, kirim permintaan melalui CloudFront, yang menandatanganinya atas nama Anda (lihatthe section called “CloudFront integrasi”).

CloudFront integrasi

Anda dapat menggunakan Amazon CloudFront Origin Access Control (OAC) untuk menempatkan CloudFront distribusi di depan titik `IamAuth` akhir. Dengan OAC, CloudFront tandatangani setiap permintaan asal atas nama pemirsa, sehingga penelepon tidak menandatangani permintaan sendiri. CloudFront tidak menggunakan `Authorization` header untuk menandatangani permintaan asal, sehingga dapat meneruskan `Authorization` header Anda ke fungsi.

Lambda menghapus `Authorization` nilai apa pun yang merupakan AWS tanda tangan (nilai yang dimulai dengan`AWS4-`). Ini memungkinkan Anda menyimpan otentikasi aplikasi Anda (misalnya,

JWT atau cookie) sambil CloudFront menandatangani setiap permintaan asal. Untuk melakukan ini, konfigurasikan distribusi untuk meneruskan `Authorization` header. Untuk informasi selengkapnya, lihat Membatasi akses ke AWS asal.

Domain endpoint tetap dapat dijangkau di internet, dan Lambda juga menerima permintaan langsung yang ditandatangani dari prinsipal mana pun yang diizinkan oleh kebijakan Anda.

Ini membutuhkan jenis otentikasi titik akhir. `IamAuth` Buat kontrol akses asal dengan tipe asal `lambda` dan perilaku penandatangananalways-amz-auth. CloudFront menolak perilaku penandatanganan lain untuk asal Fungsi Web.

> **Note**
>
> Lambda tidak mengautentikasi permintaan ke `ApplicationManaged` titik akhir atau mengevaluasi kebijakan berbasis sumber daya Fungsi Web untuk permintaan tersebut. Titik akhir seperti itu tetap dapat dijangkau publik di domainnya sendiri, bahkan di belakang. CloudFront

Untuk memungkinkan CloudFront untuk memanggil titik akhir, lampirkan kebijakan berbasis sumber daya ke Fungsi Web yang memberikan prinsip layanan () tindakan. CloudFront `cloudfront.amazonaws.com` `lambda:InvokeWebFunctionEndpoint` Gunakan `aws:SourceArn` kondisi untuk hanya mengizinkan distribusi spesifik Anda:

Example Resource-based Kebijakan untuk CloudFront OAC

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowCloudFrontOACAccess",
            "Effect": "Allow",
            "Principal": {
                "Service": "cloudfront.amazonaws.com"
            },
            "Action": "lambda:InvokeWebFunctionEndpoint",
            "Resource": "arn:aws:lambda:us-east-1:111122223333:web-function/my-
function/endpoint/prod",
            "Condition": {
                "StringEquals": {
```

```
                    "aws:SourceArn": "arn:aws:cloudfront::111122223333:distribution/
<distribution-id>"
                }
            }
        }
    ]
}
```

Menggunakan jenis **`ApplicationManaged`**otentikasi

Ketika jenis otentikasi endpoint adalah`ApplicationManaged`, siapa pun dengan domain endpoint dapat memanggil fungsi Anda. Lambda tidak melakukan otentikasi atau otorisasi.

Pilih `ApplicationManaged` kapan titik akhir Anda dapat dijangkau oleh publik, misalnya, untuk melayani permintaan langsung dari browser web. Anda juga dapat memilihnya saat menangani otentikasi dalam kode Anda sendiri, seperti OAuth 2.0, OpenID Connect, atau skema token khusus.

Dengan jenis otentikasi ini:

- Lambda memungkinkan semua penelepon untuk memanggil titik akhir. Itu tidak mengevaluasi kebijakan berbasis sumber daya untuk permintaan ini.
- Lambda meneruskan `Authorization` header pemanggil ke fungsi Anda, sehingga kode Anda dapat memvalidasinya. Jika header berisi tanda tangan AWS Signature Version 4 (SigV4), Lambda menghapusnya, bersama dengan `X-Amz-Security-Token` header apa pun, sehingga AWS kredensi tidak diteruskan ke kode fungsi Anda.
- Fungsi Anda bertanggung jawab atas semua otentikasi dan otorisasi.

> **Note**
>
> Untuk `ApplicationManaged` titik akhir, Lambda tidak mengevaluasi kebijakan berbasis sumber daya Fungsi Web untuk permintaan pemanggilan. Anda tidak dapat menggunakan kebijakan berbasis sumber daya untuk membatasi siapa yang memanggil titik akhir. `ApplicationManaged` Untuk membatasi pemanggilan, gunakan sebagai gantinya. `IamAuth` Kebijakan berbasis sumber daya masih berlaku untuk tindakan manajemen. Misalnya, dapat memberikan izin akun lain untuk memperbarui titik akhir. Untuk informasi selengkapnya, lihat the section called “Resource-based kebijakan”.

Tata kelola dan kontrol akses

Anda dapat mengontrol siapa yang dapat mengonfigurasi titik akhir.

Lambda mendukung tindakan IAM berikut untuk mengelola titik akhir:

- `lambda:CreateWebFunction`— Buat Fungsi Web dan konfigurasikan titik akhir awalnya, termasuk miliknya`AuthType`.
- `lambda:CreateWebFunctionEndpoint`— Buat titik akhir dan atur. `AuthType`
- `lambda:UpdateWebFunctionEndpoint`— Perbarui titik akhir, termasuk. `AuthType`
- `lambda:GetWebFunctionEndpoint`— Lihat konfigurasi titik akhir.
- `lambda:ListWebFunctionEndpoints`— Daftar titik akhir dari Fungsi Web.
- `lambda:DeleteWebFunctionEndpoint`— Hapus titik akhir.

Untuk mengizinkan atau menolak tindakan berdasarkan jenis otentikasi, gunakan kunci kondisi. `lambda:Request/WebFunctionAuthType`berlaku untuk`CreateWebFunction`,`CreateWebFunctionEndpoint`, dan`UpdateWebFunctionEndpoint`. `lambda:Resource/WebFunctionAuthType`berlaku untuk `InvokeWebFunctionEndpoint` (pada `IamAuth` titik akhir),, `UpdateWebFunctionEndpointGetWebFunctionEndpoint`, dan`DeleteWebFunctionEndpoint`. Lihat perinciannya di the section called “Kunci syarat”.

Membatasi titik akhir publik dalam suatu organisasi

Jika Anda mengelola AWS Organisasi dan ingin mencegah pengembang membuat titik akhir publik, gunakan kebijakan kontrol layanan (SCP) yang menolak membuat atau memperbarui titik akhir dengan jenis otentikasi. `ApplicationManaged` Gunakan tombol `lambda:Request/WebFunctionAuthType` kondisi, yang mencerminkan jenis otentikasi dalam permintaan:

Example SCP menyangkal titik akhir publik

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "lambda:CreateWebFunction",
                "lambda:CreateWebFunctionEndpoint",
```

```
                "lambda:UpdateWebFunctionEndpoint"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "lambda:Request/WebFunctionAuthType": "ApplicationManaged"
                }
            }
        }
    ]
}
```

Kebijakan termasuk `lambda:CreateWebFunction` karena operasi tersebut membuat titik akhir awal dari permintaan. `endpointConfig` Dengan kebijakan ini, `aws lambda-web deploy` gagal dengan kesalahan akses ditolak kecuali pengembang menambahkan`--auth-type IamAuth`. Secara default, perintah membuat `ApplicationManaged` titik akhir.

Kunci syarat

Untuk kontrol akses berbutir halus, Lambda mendukung kunci konteks kondisi berikut:

- `lambda:Request/WebFunctionAuthType`— Jenis otentikasi yang ditentukan dalam permintaan. Tersedia selama`CreateWebFunction`,`CreateWebFunctionEndpoint`, dan`UpdateWebFunctionEndpoint`. Gunakan untuk mengontrol jenis otentikasi mana yang dapat dikonfigurasi.
- `lambda:Resource/WebFunctionAuthType`— Jenis otentikasi yang saat ini dikonfigurasi pada titik akhir, dengan nilai `IamAuth` dan. `ApplicationManaged` Tersedia selama`InvokeWebFunctionEndpoint`,`UpdateWebFunctionEndpoint`,`DeleteWebFunctionEndpo` dan`GetWebFunctionEndpoint`. Untuk`InvokeWebFunctionEndpoint`, Lambda mengevaluasi kunci hanya pada titik akhir, karena permintaan ke `IamAuth` titik akhir tidak diotorisasi dengan `ApplicationManaged` IAM.
- `aws:ResourceTag/${TagKey}`— Memfilter akses dengan tag pada Fungsi Web. Titik akhir dan revisi mewarisi tag fungsi induk, sehingga Anda dapat menggunakan ABAC untuk mengontrol akses pemanggilan.

Example Izinkan pemanggilan hanya pada titik akhir dari fungsi yang ditandai

```
{
    "Version": "2012-10-17",
```

```
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "lambda:InvokeWebFunctionEndpoint",
            "Resource": "arn:aws:lambda:us-east-1:111122223333:web-function/*/endpoint/
prod",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/Environment": "Production"
                }
            }
        }
    ]
}
```

Enkripsi saat istirahat untuk Fungsi Web

AWS Lambda mengenkripsi data untuk Fungsi Web Lambda Anda saat istirahat di sisi server. Halaman ini menjelaskan opsi enkripsi yang tersedia untuk Anda dan cara mengonfigurasi kunci yang dikelola pelanggan.

Opsi untuk enkripsi saat istirahat

Enkripsi saat istirahat selalu aktif. Lambda mengenkripsi artefak kode Fungsi Web dan variabel lingkungan di sisi server, dan Anda tidak dapat mematikan enkripsi ini. Secara default, Lambda menggunakan file. Kunci milik AWS Anda dapat mengonfigurasi kunci yang dikelola pelanggan sebagai gantinya.

Anda memiliki dua opsi untuk AWS KMS kunci yang melindungi data Anda:

Kunci milik AWS (default)

Secara default, Lambda mengenkripsi data Anda dengan file. Kunci milik AWS AWS memiliki dan mengelola kunci ini. Anda tidak dapat melihat, mengelola, atau mengaudit penggunaannya. Jika perilaku default ini sesuai dengan kebutuhan Anda, tidak diperlukan pengaturan tambahan.

Kunci yang dikelola pelanggan

Anda dapat secara opsional mengonfigurasi kunci yang dikelola AWS KMS pelanggan Anda sendiri. Gunakan kunci yang dikelola pelanggan saat Anda perlu mengontrol rotasi kunci, menggunakan kunci audit CloudTrail, mengelola akses kunci melalui kebijakan utama, atau memenuhi persyaratan kepatuhan organisasi. Dengan kunci yang dikelola pelanggan, hanya

prinsipal yang Anda otorisasi yang dapat mengakses kode terenkripsi dan variabel lingkungan. Kunci yang dikelola pelanggan dikenakan AWS KMS biaya. Untuk informasi selengkapnya, lihat harga AWS Key Management Service.

Mengenkripsi data saat istirahat menggunakan kunci yang dikelola pelanggan

Bagian berikut menjelaskan bagaimana Fungsi Web menggunakan kunci yang dikelola pelanggan, cara mengonfigurasi kunci dan izinnya, dan cara mengurangi akses ke sana.

Bagaimana Fungsi Web menggunakan kunci yang dikelola pelanggan

Saat Anda menentukan kunci terkelola pelanggan, Lambda menggunakan enkripsi amplop. Ini berarti Lambda mengenkripsi data revisi dengan kunci data, dan kunci KMS Anda mengenkripsi kunci data tersebut. Lambda mendapatkan kunci data dari AWS KMS dan mendekripsi dengan kunci KMS Anda ketika perlu membaca data. Hanya prinsipal dengan akses ke kunci KMS yang dapat mengakses kode revisi dan variabel lingkungan.

Lambda mengenkripsi sumber daya Fungsi Web berikut dengan kunci terkelola pelanggan Anda:

Sumber daya	Apa yang dienkripsi
Revisi	Artefak kode yang dibuat dari paket penerapan Amazon S3 Anda, dan variabel lingkungan revisi.

Lambda tidak menggunakan AWS KMS hibah untuk Fungsi Web. Akses diotorisasi melalui kebijakan kunci dan izin IAM kepala pemanggil.

Lambda memanggil AWS KMS sambil membuat dan meluncurkan sumber daya revisi dan dapat menyimpan materi yang didekripsi di lingkungan eksekusi aktif. Mencabut akses kunci bukanlah mekanisme penahanan lalu lintas langsung karena lingkungan yang ada dapat terus menggunakan materi yang di-cache untuk periode yang tidak ditentukan.

Setelah perubahan kebijakan kunci, uji perilaku yang diharapkan dan pantau revisi dan kesehatan titik akhir. Untuk penahanan darurat, hentikan perutean lalu lintas ke revisi yang terpengaruh atau nonaktifkan titik akhir daripada hanya mengandalkan pencabutan kunci.

Lambda menetapkan konteks enkripsi berikut pada semua AWS KMS operasi untuk Fungsi Web:

- `aws:lambda:WebFunctionRevisionArn`— Nama Sumber Daya Amazon (ARN) dari revisi Fungsi Web.

Anda dapat menggunakan konteks enkripsi ini dalam kebijakan utama untuk cakupan akses ke Fungsi Web tertentu atau revisi. Untuk informasi selengkapnya, lihat the section called “Mencakup akses ke kunci KMS Anda”.

Mengonfigurasi kunci terkelola pelanggan

AWS KMS Kunci harus berupa kunci enkripsi simetris (spesifikasi kunci`SYMMETRIC_DEFAULT`, penggunaan `ENCRYPT_DECRYPT` kunci). Anda tidak dapat menggunakan tombol asimetris.

Fungsi Web mendukung tombol Multi-wilayah. Jenis kunci yang Anda butuhkan tergantung pada konfigurasi titik akhir Anda:

- HomeRegion endpoint — Anda dapat menggunakan kunci Single-region atau kunci Multi-region.
- MultiRegion atau PerRegion titik akhir - Anda harus menggunakan kunci Multi-wilayah dengan replika di setiap titik akhir Wilayah AWS yang disajikan. Jika Wilayah tidak memiliki replika kunci, replikasi ke Wilayah tersebut gagal.

Untuk informasi tentang membuat kunci Multi-wilayah, lihat Membuat kunci utama Multi-wilayah di Panduan AWS Key Management Service Pengembang.

Karena AWS KMS tidak menyalin atau menyinkronkan kebijakan kunci antara kunci Multi-wilayah terkait, Anda harus mengonfigurasi setiap kebijakan kunci secara terpisah. Konfigurasikan kebijakan kunci pada kunci utama dan pada setiap kunci replika. Minimal, kebijakan setiap kunci replika harus mengizinkan kepala layanan Lambda untuk menelepon. `kms:Decrypt` Izin ini memungkinkan Lambda untuk mendekripsi kunci data di Wilayah tersebut. Untuk informasi selengkapnya, lihat Mengotorisasi penggunaan kunci Multi-wilayah di Panduan AWS Key Management Service Pengembang.

> **Note**
>
> Saat Anda membuat kunci replika Multi-wilayah, kunci tersebut dapat memakan waktu dari beberapa detik hingga beberapa menit untuk tersedia di semua Wilayah. Konfirmasikan bahwa kunci replika tersedia di setiap Wilayah target sebelum Anda membuat revisi atau titik akhir.

Mengonfigurasi izin untuk menggunakan kunci terkelola pelanggan

Untuk membuat revisi dengan kunci yang dikelola pelanggan, Anda memerlukan izin berikut pada AWS KMS kunci:

- `kms:DescribeKey`— Diperlukan saat Anda membuat Fungsi Web atau revisi dengan kunci.
- `kms:GenerateDataKey`— Lambda memanggil operasi ini untuk menghasilkan kunci data yang mengenkripsi artefak kode revisi dan variabel lingkungan.
- `kms:Decrypt`— Lambda memanggil operasi ini untuk memvalidasi bahwa kunci dapat digunakan saat Anda membuat revisi. Prinsipal yang membaca revisi dengan `GetWebFunctionRevision` juga membutuhkan kunci untuk melihat `kms:Decrypt` variabel lingkungannya.

Permintaan pembuatan gagal jika kunci hilang, dinonaktifkan, bukan kunci enkripsi simetris, atau tidak dapat diakses dengan izin yang diperlukan.

> **Note**
>
> Anda tidak perlu `kms:Encrypt` izin. Lambda menggunakan enkripsi amplop dengan`GenerateDataKey`, yang mengembalikan teks biasa dan salinan kunci data terenkripsi tanpa memerlukan panggilan enkripsi terpisah.

Kebijakan AWS KMS kunci juga harus mengizinkan Lambda service principal (`lambda.amazonaws.com`) untuk menelepon. `kms:Decrypt` Lambda menggunakan izin ini untuk mendekripsi kunci data saat meluncurkan lingkungan eksekusi untuk revisi. Prinsipal layanan tidak perlu`kms:DescribeKey`. Prinsipal yang membuat Fungsi Web atau revisi membutuhkan izin itu.

Contoh kebijakan kunci

Kebijakan kunci berikut berisi tiga pernyataan yang dibutuhkan Web Functions, masing-masing dicakup dengan kondisi yang dijelaskan dalam. the section called “Mencakup akses ke kunci KMS Anda” Pernyataan pertama memberi kepala layanan Lambda izin yang dibutuhkan saat meluncurkan lingkungan eksekusi. Yang kedua dan ketiga memberikan prinsipal yang membuat revisi izin yang dibutuhkan pada waktu pembuatan.

`kms:DescribeKey`ada dalam pernyataan terpisah karena ini adalah operasi metadata yang tidak membawa konteks enkripsi. Kondisi konteks enkripsi tidak akan pernah cocok dengan

`DescribeKey` permintaan, sehingga kondisi tersebut hanya berlaku untuk pernyataan yang memberikan `kms:GenerateDataKey` dan`kms:Decrypt`.

Contoh ini hanya menunjukkan pernyataan yang dibutuhkan Lambda untuk Fungsi Web. Tambahkan pernyataan ini ke kebijakan kunci yang ada daripada menggantinya, dan simpan pernyataan yang memberikan akses administrator Anda untuk mengelola kunci.

Example Kebijakan kunci untuk enkripsi Fungsi Web

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Allow Lambda to decrypt the data key for Web Function revisions",
            "Effect": "Allow",
            "Principal": { "Service": "lambda.amazonaws.com" },
            "Action": "kms:Decrypt",
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "kms:EncryptionContext:aws:lambda:WebFunctionRevisionArn":
 "arn:aws:lambda:us-east-1:111122223333:web-function/my-function/revision/*",
                    "aws:SourceArn": "arn:aws:lambda:us-east-1:111122223333:web-
function/my-function/revision/*"
                }
            }
        },
        {
            "Sid": "Allow the revision creator to generate and validate data keys",
            "Effect": "Allow",
            "Principal": { "AWS":
 "arn:aws:iam::111122223333:role/MyWebFunctionDeployRole" },
            "Action": [
                "kms:GenerateDataKey",
                "kms:Decrypt"
            ],
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "kms:EncryptionContext:aws:lambda:WebFunctionRevisionArn":
 "arn:aws:lambda:us-east-1:111122223333:web-function/my-function/revision/*"
                },
                "StringEquals": {
```

```
                    "kms:ViaService": "lambda.us-east-1.amazonaws.com"
                }
            }
        },
        {
            "Sid": "Allow the revision creator to describe the key",
            "Effect": "Allow",
            "Principal": { "AWS":
 "arn:aws:iam::111122223333:role/MyWebFunctionDeployRole" },
            "Action": "kms:DescribeKey",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "kms:ViaService": "lambda.us-east-1.amazonaws.com"
                }
            }
        }
    ]
}
```

Untuk informasi selengkapnya tentang kebijakan AWS KMS utama, lihat Kebijakan utama AWS KMS di Panduan AWS Key Management Service Pengembang.

Membuat revisi Fungsi Web dengan kunci yang dikelola pelanggan

Untuk mengenkripsi revisi dengan kunci yang dikelola pelanggan, tentukan ARN kunci dalam `kmsKeyArn` parameter konfigurasi revisi. Anda dapat mengatur parameter ini saat Anda memanggil CreateWebFunctionatau CreateWebFunctionRevision. Jika Anda menghilangkan`kmsKeyArn`, Lambda mengenkripsi revisi dengan. Kunci milik AWS

Untuk membuat Fungsi Web dengan kunci yang dikelola pelanggan (AWS CLI)

- Jalankan perintah berikut untuk membuat Fungsi Web dengan kunci yang dikelola pelanggan.

```
aws lambda-web create-web-function \
    --function-name my-web-app \
    --revision-config '{
      "kmsKeyArn": "arn:aws:kms:us-east-1:111122223333:key/key-id",
      "buildConfig": {
        "codeConfig": {
          "s3Object": { "bucket": "amzn-s3-demo-bucket", "key": "my-app.zip" }
        },
        "runtimeConfig": { "runtime": "nodejs24.x" }
```

```
    },
    "serviceConfig": {
      "executionRoleArn": "arn:aws:iam::111122223333:role/my-web-function-role"
    }
  }' \
  --endpoint-config '{
    "endpointName": "my-endpoint",
    "endpointType": "HomeRegion",
    "authType": "ApplicationManaged"
  }'
```

Mengubah konfigurasi enkripsi Fungsi Web yang ada

`kmsKeyArn`Parameter ditetapkan untuk masa revisi. Karena revisi tidak dapat diubah, Anda tidak dapat mengubah kunci yang melindungi revisi yang ada. Untuk mengubah konfigurasi enkripsi, buat revisi baru yang menentukan kunci baru, lalu rute lalu lintas ke revisi tersebut.

Fungsi Web mendukung transisi kunci berikut:

- Kunci milik AWS to customer managed key — Buat revisi baru yang menentukan`kmsKeyArn`.
- Satu pelanggan mengelola kunci ke kunci terkelola pelanggan yang berbeda — Buat revisi baru yang menentukan ARN kunci baru.
- Kunci yang dikelola pelanggan untuk Kunci milik AWS — Buat revisi baru tanpa`kmsKeyArn`. Lambda kemudian melindungi revisi itu dengan Kunci milik AWS, yang merupakan default.

Lambda tidak mengenkripsi ulang revisi yang ada saat Anda membuat revisi baru dengan kunci yang berbeda. Setiap revisi tetap dienkripsi dengan kunci yang Anda tentukan saat Anda membuatnya. Ini berarti bahwa selama revisi sebelumnya masih dapat menerima lalu lintas, kunci aslinya harus tetap diaktifkan dan dapat diakses oleh kepala layanan Lambda. Sebelum Anda menonaktifkan atau menghapus kunci, konfirmasikan bahwa tidak ada revisi yang dienkripsi dengan kunci tersebut yang masih direferensikan oleh bobot revisi titik akhir.

Untuk membuat revisi baru dengan kunci terkelola pelanggan yang berbeda (AWS CLI)

1. Jalankan perintah berikut untuk membuat revisi yang menggunakan kunci yang berbeda.

```
aws lambda-web create-web-function-revision \
    --function-name my-web-app \
    --kms-key-arn arn:aws:kms:us-east-1:111122223333:key/new-key-id \
```

```
    --build-config '{
      "codeConfig": { "s3Object": { "bucket": "my-bucket", "key": "my-app-
v2.zip" } },
      "runtimeConfig": { "runtime": "nodejs24.x" }
    }' \
    --service-config '{
      "executionRoleArn": "arn:aws:iam::111122223333:role/my-web-function-role"
    }'
```

2. Perbarui bobot revisi titik akhir untuk merutekan lalu lintas ke revisi baru. Untuk informasi selengkapnya, lihat the section called “Pola deployment”.

Mencakup akses ke kunci KMS Anda

Kebijakan kunci dalam the section called “Contoh kebijakan kunci” membatasi akses dengan kunci kondisi berikut. Masing-masing membatasi bagaimana kunci dapat digunakan, sehingga akses ke kunci tidak lebih luas dari kebutuhan Fungsi Web.

Konteks enkripsi

Lambda menetapkan konteks enkripsi `aws:lambda:WebFunctionRevisionArn` pada setiap AWS KMS permintaan revisi. Gunakan `StringLike` kondisi `kms:EncryptionContext:aws:lambda:WebFunctionRevisionArn` untuk membatasi kunci revisi Fungsi Web tertentu. Terapkan kondisi ini pada pernyataan pembuat dan pernyataan utama layanan Lambda.

Jangan terapkan kondisi ini pada pernyataan yang memberikan. `kms:DescribeKey` Operasi itu tidak membawa konteks enkripsi, sehingga kondisinya akan menolaknya.

Contoh berikut menunjukkan pernyataan pembuat dengan kondisi konteks enkripsi.

```
{
    "Sid": "Allow the revision creator to generate and validate data keys",
    "Effect": "Allow",
    "Principal": { "AWS": "arn:aws:iam::111122223333:role/MyWebFunctionDeployRole" },
    "Action": [
        "kms:GenerateDataKey",
        "kms:Decrypt"
    ],
    "Resource": "*",
    "Condition": {
        "StringLike": {
```

```
            "kms:EncryptionContext:aws:lambda:WebFunctionRevisionArn":
 "arn:aws:lambda:us-east-1:111122223333:web-function/my-function/revision/*"
        }
    }
}
```

kms: kondisi ViaService

`kms:ViaService`Kondisi membatasi kunci sehingga hanya permintaan yang dibuat Lambda atas nama Anda yang dapat menggunakannya. Nilainya adalah Region-specific:`lambda.us-east-1.amazonaws.com`.

Dengan kondisi ini, AWS KMS menolak panggilan langsung yang dilakukan dengan kredensil yang sama (misalnya, menjalankan `aws kms generate-data-key` sendiri), tetapi membuat Fungsi Web berhasil karena Lambda membuat panggilan untuk Anda. AWS KMS

AWS KMS tidak membagikan kebijakan utama antara kunci Multi-wilayah terkait. Setiap kunci replika memiliki kebijakannya sendiri. Bila Anda menggunakan `kms:ViaService` kondisi dengan kunci Multi-region, gunakan nama layanan Regional yang cocok dengan Region masing-masing kunci (misalnya, `lambda.us-east-1.amazonaws.com` dalam kebijakan kunci us-east-1 dan `lambda.us-west-2.amazonaws.com` dalam kebijakan kunci replika us-west-2). Anda juga dapat mencantumkan beberapa nilai Regional dalam satu kondisi karena `StringEquals` menerima array nilai.

Contoh berikut menunjukkan pernyataan pembuat dengan `kms:ViaService` kondisi dan kondisi konteks enkripsi.

```
{
    "Sid": "Allow the revision creator to generate and validate data keys",
    "Effect": "Allow",
    "Principal": { "AWS": "arn:aws:iam::111122223333:role/MyWebFunctionDeployRole" },
    "Action": [
        "kms:GenerateDataKey",
        "kms:Decrypt"
    ],
    "Resource": "*",
    "Condition": {
        "StringLike": {
            "kms:EncryptionContext:aws:lambda:WebFunctionRevisionArn":
 "arn:aws:lambda:us-east-1:111122223333:web-function/my-function/revision/*"
        },
        "StringEquals": {
```

```
            "kms:ViaService": "lambda.us-east-1.amazonaws.com"
        }
    }
}
```

Perlindungan wakil yang bingung

Masalah wakil yang membingungkan terjadi ketika entitas tanpa izin untuk melakukan tindakan memaksa entitas yang lebih istimewa untuk melakukannya. Karena prinsipal layanan Lambda dapat menggunakan kunci yang dikelola pelanggan Anda, batasi akses ke revisi Fungsi Web yang Anda miliki. Untuk informasi selengkapnya, lihat Masalah confused deputy di Panduan Pengguna IAM.

Ketika Lambda memanggil AWS KMS dengan kredensi layanannya sendiri untuk meluncurkan lingkungan eksekusi, ia mengirimkan ARN revisi sebagai ARN sumber dan ID Anda sebagai akun sumber. Akun AWS Gunakan kunci SourceArn kondisi aws: pada pernyataan utama layanan Lambda untuk membatasi kunci revisi Fungsi Web tertentu.

```
{
    "Sid": "Allow Lambda to decrypt the data key for Web Function revisions",
    "Effect": "Allow",
    "Principal": { "Service": "lambda.amazonaws.com" },
    "Action": "kms:Decrypt",
    "Resource": "*",
    "Condition": {
        "StringLike": {
            "aws:SourceArn": "arn:aws:lambda:us-east-1:111122223333:web-function/my-
function/revision/*"
        }
    }
}
```

Untuk membatasi kunci ke akun Anda alih-alih ke Fungsi Web tertentu, gunakan kunci SourceAccount kondisi aws: dengan Akun AWS ID Anda.

Terapkan ketentuan ini pada pernyataan utama layanan Lambda. `kms:ViaService`Kondisi yang dijelaskan dalam the section called “kms: kondisi ViaService” membatasi layanan mana yang dapat menggunakan kunci, tetapi tidak memberikan perlindungan wakil yang membingungkan.

Perilaku kesalahan

Lambda memvalidasi AWS KMS kunci saat Anda membuat revisi. Kesalahan berikut dapat terjadi pada waktu pembuatan:

- Kesalahan validasi — Kunci ARN tidak valid, kunci tidak ada, kunci dinonaktifkan, atau kuncinya bukan kunci enkripsi simetris.
- Kesalahan akses ditolak — Prinsipal panggilan atau kepala layanan Lambda tidak memiliki AWS KMS izin yang diperlukan pada kunci.

> ⚠ Jangan menonaktifkan atau mencabut akses ke kunci revisi aktif
>
> Jika Anda menonaktifkan AWS KMS kunci atau mencabut akses kepala layanan Lambda setelah revisi`Active`, Lambda tidak dapat lagi mendekripsi kunci data. Ini mencegah Lambda meluncurkan lingkungan eksekusi baru untuk revisi itu, yang merusak pemanggilan. Juga jangan hapus kunci replika di Wilayah tempat revisi masih digunakan. Ini memiliki efek yang sama (Lambda tidak dapat meluncurkan lingkungan eksekusi baru di Wilayah itu). Untuk memulihkan layanan, buat revisi baru dengan kunci yang dapat diakses, atau aktifkan kembali kunci asli dan pulihkan kebijakan kunci.

Enkripsi dengan MultiRegion dan titik PerRegion akhir

Untuk Fungsi Web dengan MultiRegion atau PerRegion titik akhir, Lambda memeriksa ketersediaan kunci di setiap Wilayah selama replikasi. Jika replika kunci Multi-wilayah hilang atau tidak dapat diakses di Wilayah target, replikasi hanya gagal untuk Wilayah tersebut. Daerah lain terus beroperasi secara normal.

Anda dapat melihat setiap status Wilayah dan memperbarui status dengan meneleponGetWebFunctionEndpoint. Setiap entri Wilayah `regionalEndpoints` berisi `state` dan `stateReason` (status penyebaran Wilayah) `updateStatus` dan dan `updateStatusReason` (hasil pembaruan terbaru). Perilaku berbeda tergantung pada apakah titik akhir sedang dibuat atau diperbarui:

- Selama pembuatan titik akhir — Entri Wilayah `regionalEndpoints` menunjukkan a `state` `Failed` dengan a `stateReason` yang menjelaskan masalah utama. Titik akhir keseluruhan `state` juga ditampilkan`Failed`, dan `stateReason` mengarahkan Anda ke entri Regional.
- Selama pembaruan titik akhir - Wilayah `updateStatus` menunjukkan `Failed` dengan `updateStatusReason` yang menjelaskan masalah utama. Titik akhir keseluruhan `updateStatus` juga ditampilkan`Failed`, dan `updateStatusReason` mengarahkan Anda ke entri Regional.

Untuk mengatasi Wilayah yang gagal, buat replika kunci di Wilayah tersebut atau pulihkan akses ke replika yang ada, lalu perbarui titik akhir untuk mencoba lagi replikasi.

Memantau interaksi Fungsi Web dengan AWS KMS

Anda dapat menggunakan CloudTrail untuk melacak AWS KMS permintaan yang dibuat Lambda atas nama Anda saat Anda menggunakan kunci yang dikelola pelanggan dengan Fungsi Web. Untuk informasi tentang menemukan AWS KMS peristiwa di CloudTrail log Anda, lihat Mencari CloudTrail log untuk panggilan AWS KMS API di Panduan AWS Key Management Service Pengembang.

Tabel berikut mencantumkan AWS KMS peristiwa yang dihasilkan Web Functions dan nilai yang dapat Anda audit di setiap peristiwa.

eventName	Ketika itu terjadi	Nilai untuk diaudit
`DescribeKey`	Saat Anda membuat Fungsi Web atau revisi dengan kunci yang dikelola pelanggan.	`userIdentity` adalah prinsip IAM yang menciptakan revisi. Acara ini tidak memiliki konteks enkripsi, karena `DescribeKey` merupakan operasi metadata.
`GenerateDataKey`	Saat Anda membuat Fungsi Web atau revisi. Lambda menghasilkan kunci data yang mengenkripsi artefak kode revisi dan variabel lingkungan.	`userIdentity` adalah prinsip IAM yang menciptakan revisi. `encryptionContext` berisi`aws:lambda:WebFunctionRevisionArn` .
`Decrypt`	Saat Anda membuat Fungsi Web atau revisi, Lambda memvalidasi akses kunci. Lambda juga memanggil operasi ini ketika menyebarkan revisi ke Region replika, ketika meluncurkan lingkungan eksekusi baru, dan ketika pemanggil membaca variabel lingkungan revisi dengan. `GetWebFunctionRevision`	Untuk panggilan waktu pembuatan , `userIdentity` adalah prinsip IAM yang membuat revisi. Lambda juga melakukan `Decrypt` panggilan dry-run `lambda.amazonaws.com` untuk mengonfirmasi bahwa prinsipal layanan dapat menggunakan kunci tersebut. Untuk panggilan runtime, `userIdentity` adalah Lambda service principal `lambda.amazonaws.com` (). Untuk`GetWebFun`

eventName	Ketika itu terjadi	Nilai untuk diaudit
		`ctionRevision` , Lambda memanggil `Decrypt` dengan kredensi penelepon. `encryptio nContext` berisi`aws:lambd a:WebFunctionRevisionArn` .

`Decrypt`Peristiwa `GenerateDataKey` dan termasuk konteks enkripsi `aws:lambda:WebFunctionRevisionArn` dengan ARN revisi. Gunakan nilai ini untuk mengidentifikasi AWS KMS permintaan untuk revisi Fungsi Web tertentu.

Memantau Fungsi Web Lambda

AWS Lambda Fungsi Web mempublikasikan metrik dan log ke Amazon CloudWatch. Gunakan mereka untuk mengidentifikasi kemacetan kinerja, kesalahan debug, dan setel alarm yang mengingatkan Anda saat tingkat kesalahan melonjak selama penerapan kenari atau pergeseran lalu lintas.

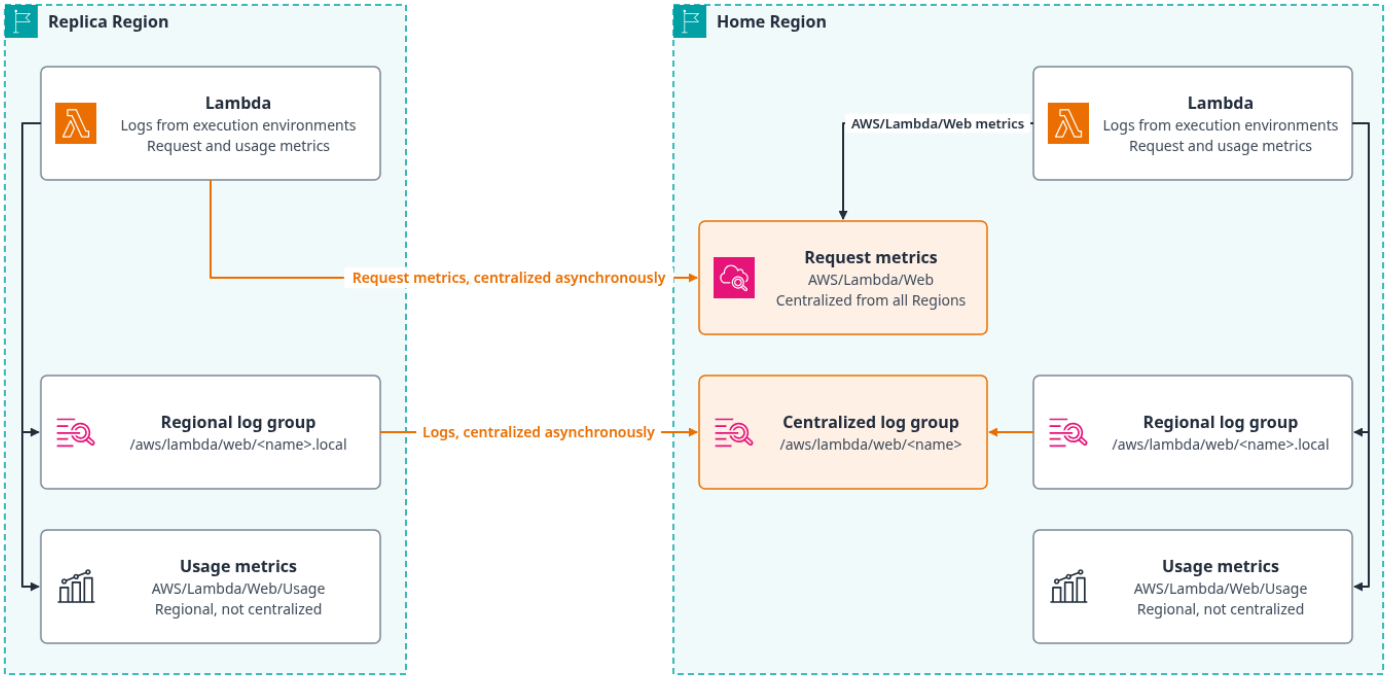
Untuk Fungsi Web yang menyajikan permintaan di lebih dari satu Wilayah AWS, Lambda mengumpulkan metrik dan log di setiap Wilayah tempat permintaan dijalankan. Ini memusatkan metrik permintaan dan log ke Wilayah rumah Fungsi Web, sehingga Anda dapat memantau seluruh aplikasi dari satu tempat. Metrik penggunaan dan pembatasan tetap berada di Wilayah tempat metrik tersebut dipancarkan, karena kuota yang dideskripsikan berlaku per Wilayah.

Setiap jenis telemetri menjawab pertanyaan yang berbeda:

- Metrik — Lacak kinerja Fungsi Web Anda dari waktu ke waktu, dan alarm tentang masalah. Lihat perinciannya di the section called “Metrik-metrik”.
- Log — Selidiki permintaan dan kesalahan individu. Lihat perinciannya di the section called “Beberapa catatan”.
- Metrik kustom — Publikasikan metrik aplikasi Anda sendiri dengan menulis format metrik yang disematkan ke log Anda. Lihat perinciannya di the section called “Metrik-metrik kustom”.
- CloudTrail— Rekam panggilan API untuk Fungsi Web Anda, termasuk pemanggilan, yang dicatat sebagai CloudTrail peristiwa data. Lihat perinciannya di the section called “CloudTrail log”.

Untuk memecahkan masalah di Wilayah tempat permintaan dijalankan, gunakan grup log Regional/`aws/lambda/web/`*`function-name`*`.local`, tempat lingkungan eksekusi menulis log. Lambda memusatkan log secara asinkron ke grup `/aws/lambda/web/`*`function-name`* log di Wilayah asal. Untuk informasi tentang Wilayah dan replikasi asal, lihatthe section called “Multi-Region penyebaran”.

Diagram berikut menunjukkan bagaimana Lambda menulis dan memusatkan telemetri di seluruh Wilayah. Metrik permintaan di `AWS/Lambda/Web` namespace dan log dipusatkan di Wilayah beranda. Metrik penggunaan di `AWS/Lambda/Web/Usage` namespace tetap ada di setiap Wilayah.



Topik

- Penagihan telemetri
- Pertimbangan telemetri
- Menggunakan CloudWatch metrik dengan Fungsi Web Lambda
- Log Fungsi Web di CloudWatch Log Amazon
- Menerbitkan metrik kustom dengan format metrik yang disematkan
- Logging panggilan API Fungsi Web menggunakan AWS CloudTrail

Penagihan telemetri

Telemetri untuk Fungsi Web ditagih sebagai berikut:

Telemetri	Penagihan
Metrik diterbitkan oleh Lambda	Bebas. Lambda menerbitkan metrik di `AWS/Lambda/Web/Usage` ruang nama `AWS/Lambda/Web` dan tanpa biaya kepada Anda.
Log	Anda membayar biaya CloudWatch Log Amazon standar untuk menulis dan menyimpan log di grup `.local` log di setiap Wilayah. Anda juga membayar untuk menyimpan salinan di grup log terpusat di Wilayah asal. Lambda menyalin setiap peristiwa log ke Wilayah asal tanpa biaya. Sentralisasi pertama dari acara log gratis.
Metrik-metrik kustom	Metrik khusus yang Anda terbitkan menggunakan format metrik yang disematkan dikenakan biaya konsumsi CloudWatch Log Amazon dan CloudWatch biaya metrik khusus. Sentralisasi pertama dari acara log gratis. Untuk informasi selengkapnya, lihat the section called “Metrik-metrik kustom”.

Untuk harga saat ini, lihat CloudWatchharga Amazon.

Pertimbangan telemetri

Tinjau perilaku berikut sebelum Anda membuat dasbor, alarm, atau otomatisasi pada telemetri Fungsi Web.

AWS X-Ray melacak

Jika permintaan menyertakan `X-Amzn-Trace-Id` header, Lambda meneruskannya ke aplikasi Anda tidak berubah. Lambda tidak menambahkan header jejak ke permintaan yang tidak menyertakannya, dan Fungsi Web tidak membuat segmen layanan Lambda. Karena penelepon mengatur header ini, perlakukan nilainya sebagai masukan klien. Untuk merekam jejak, kirim data jejak dari kode Anda langsung ke X-Ray, misalnya dengan AWS Distro for OpenTelemetry, dan berikan izin peran eksekusi Anda untuk X-Ray menulis.

Latensi sentralisasi

Metrik dan log mencapai Wilayah asal secara asinkron. Dalam kondisi mapan, Lambda biasanya memusatkan metrik dan log dalam beberapa menit. Telemetri dapat tiba nanti selama gangguan layanan.

Saat Anda membuat titik akhir yang berjalan di lebih dari satu Wilayah, Lambda menyiapkan sentralisasi metrik untuk setiap Region replika. CloudWatch dapat memakan waktu beberapa menit setelah titik akhir menjadi aktif di Wilayah untuk mulai memusatkan metriknya dari Wilayah tersebut. Metrik yang dipancarkan sebelum kemudian tetap hanya di Wilayah itu dan tidak ditambahkan ke Wilayah asal nanti.

Perhitungkan latensi ini dalam otomatisasi penerapan. Jika pipeline membuat Fungsi Web atau titik akhir dan kemudian segera memeriksa metrik atau log di Wilayah asal, berikan waktu untuk telemetri pertama tiba daripada memperlakukan ketidakhadirannya sebagai kegagalan.

Telemetri di setiap Wilayah

Lambda menulis telemetri di setiap Wilayah di mana permintaan dijalankan dan kemudian memusatkan sebagian darinya ke Wilayah asal. Apa yang dapat Anda andalkan di setiap Wilayah tergantung pada jenis telemetri:

Minta metrik (`AWS/Lambda/Web`)

Lambda juga dapat menulis metrik ini di Wilayah lain tempat permintaan dijalankan. Mereka dapat berubah atau dihapus tanpa pemberitahuan, jadi sebaiknya Anda tidak bergantung padanya dan sebagai gantinya gunakan nilai terpusat di Wilayah rumah untuk alarm, dasbor, dan otomatisasi. Nilai terpusat mencakup permintaan dari setiap Wilayah yang dilayani titik akhir.

Metrik penggunaan dan pelambatan (`AWS/Lambda/Web/Usage`)

Metrik ini hanya ada di Wilayah tempat permintaan dilayani. Alarm pada mereka di setiap Wilayah. Lihat perinciannya di the section called “Penggunaan regional dan metrik pelambatan”.

Beberapa catatan

Di setiap Wilayah, grup `.local` log hanya menyimpan log dari Wilayah tersebut. Log ini tersedia sebelum sentralisasi selesai dan tidak terpengaruh oleh penundaan sentralisasi, yang membuatnya berguna untuk men-debug masalah Regional. Grup log terpusat di Wilayah asal menyimpan log dari setiap Wilayah. Buat kueri bersama, langganan, dan kebijakan retensi pada grup log terpusat. Lihat perinciannya di the section called “Grup log”.

Gangguan layanan

Gangguan infrastruktur, seperti pemadaman Availability Zone, dapat menunda telemetri, dan beberapa metrik dan peristiwa log mungkin hilang. Jika Wilayah asal terganggu, metrik dan log terpusat di Wilayah asal dapat ditunda atau tidak tersedia. Metrik penggunaan dan grup `.local` log di Wilayah lain tidak bergantung pada sentralisasi dan terus menerima telemetri dari Wilayah tersebut.

Telemetri dipancarkan atas dasar upaya terbaik. Pada kesempatan langka, telemetri dapat dijatuhkan atau diduplikasi. Jangan memperlakukan telemetri Web Function sebagai catatan audit. Untuk mengaudit perubahan konfigurasi dan pemanggilan, gunakan. AWS CloudTrail Untuk informasi selengkapnya, lihat the section called “CloudTrail log”.

Izin yang diperlukan

Logging memerlukan izin Amazon CloudWatch Logs `logs:CreateLogGrouplogs:CreateLogStream`, dan `logs:PutLogEvents` dalam peran eksekusi Fungsi Web Anda. Tanpa mereka, Lambda tidak dapat membuat grup log atau menulis peristiwa log, dan output aplikasi Anda hilang. Untuk contoh kebijakan, lihat the section called “Peran eksekusi”.

Lambda menerbitkan metrik di `AWS/Lambda/Web/Usage` ruang nama `AWS/Lambda/Web` dan itu sendiri, sehingga mereka tidak memerlukan izin dalam peran eksekusi, termasuk. `cloudwatch:PutMetricData` Format metrik yang disematkan juga tidak memanggil `PutMetricData` secara langsung, tetapi memerlukan izin CloudWatch Log Amazon yang sama, karena Lambda mengekstrak metrik dari peristiwa log Anda.

Menggunakan CloudWatch metrik dengan Fungsi Web Lambda

AWS Lambda menerbitkan metrik Fungsi Web ke Amazon CloudWatch dalam dua ruang nama:

- `AWS/Lambda/Web`— Metrik permintaan dan respons untuk setiap titik akhir, dan metrik penggunaan sumber daya untuk lingkungan eksekusi yang melayani setiap Fungsi Web. Lambda memusatkan metrik ini ke Wilayah asal Fungsi Web, sehingga nilai di Wilayah asal mencakup lalu lintas dari setiap Wilayah yang dilayani titik akhir.
- `AWS/Lambda/Web/Usage`— Penggunaan regional dan metrik pembatasan yang menunjukkan konsumsi terhadap kuota Regional dan batas titik akhir. Metrik ini tidak terpusat. Mereka tetap berada di Wilayah tempat permintaan itu dilayani.

Lambda menerbitkan metrik permintaan, penggunaan, dan pembatasan hanya jika permintaan menghasilkan nilai untuk mereka. Metrik lingkungan eksekusi berbeda: Lambda menerbitkannya setiap menit untuk setiap lingkungan eksekusi aktif. Beberapa metrik jarang: Lambda tidak mempublikasikan titik data 0 untuk periode di mana peristiwa tidak terjadi. Misalnya, jika tidak ada permintaan yang mengembalikan respons 5xx dalam suatu periode, tidak `HTTPResponseStatusClass5XX` memiliki titik data untuk periode tersebut. Tabel berikut menjelaskan kapan Lambda memancarkan setiap metrik. Saat Anda membuat alarm pada metrik yang jarang, konfigurasikan cara alarm menangani data yang hilang. Untuk informasi selengkapnya, lihat Mengonfigurasi cara CloudWatch alarm menangani data yang hilang di CloudWatch Panduan Pengguna Amazon.

Topik

- Metrik permintaan
- Metrik lingkungan eksekusi
- Penggunaan regional dan metrik pelambatan
- Menggunakan metrik dengan fitur lain CloudWatch
- Mengatur alarm pada respons 5xx

Metrik permintaan

Metrik permintaan dipublikasikan di `AWS/Lambda/Web` namespace dengan dimensi dan`FunctionName`. `EndpointName` Mereka menggunakan `Count` unit. Gunakan `Sum` statistiknya.

Metrik	Statistik	Saat Lambda memancarkannya	Deskripsi
`ExecutedRequests`	`Sum`	Hanya untuk periode di mana setidaknya satu permintaan mencapai lingkungan eksekusi.	Jumlah permintaan yang mencapai lingkungan eksekusi. Permintaan yang dicekik atau ditolak oleh Lambda sebelum mencapai lingkungan eksekusi tidak dihitung.
`HTTPResponseStatusClass4XX`	`Sum`	Jarang. Hanya untuk periode dengan setidaknya satu respons 4xx.	Jumlah tanggapan yang dikembalikan ke penelepon dengan kode status dari 400 hingga 499. Respons dapat berasal dari aplikasi Anda atau

Metrik	Statistik	Saat Lambda memancarkannya	Deskripsi
			dari Lambda, termasuk permintaan terbatas yang menerima HTTP 429. Permintaan yang ditolak Lambda sebelum mengidentifikasi titik akhir, seperti permintaan yang salah bentuk, tidak dihitung.
`HTTPResponseStatusClass5XX`	`Sum`	Jarang. Hanya untuk periode dengan setidaknya satu respons 5xx.	Jumlah tanggapan yang dikembalikan ke penelepon dengan kode status dari 500 hingga 599. Respons dapat berasal dari aplikasi Anda atau dari Lambda. Selama gangguan layanan, metrik ini mungkin mengurangi kesalahan.

Metrik kelas status menghitung kode status yang diterima pemanggil. Mereka tidak membedakan tanggapan yang ditulis oleh aplikasi Anda dari tanggapan yang dikembalikan oleh Lambda. Jika aplikasi Anda telah mengirim status respons dan header, kegagalan selanjutnya, seperti aliran terputus atau batas waktu permintaan, dihitung di bawah kode status asli. Untuk mengidentifikasi sumber kesalahan, gunakan header `lambda-web-error-type` respons dan log Anda. Untuk menghitung kesalahan aplikasi secara terpisah, publikasikan metrik Anda sendiri. Untuk informasi selengkapnya, lihat the section called “Metrik-metrik kustom”.

Untuk memperkirakan tingkat kesalahan server, bagi `HTTPResponseStatusClass5XX` dengan `ExecutedRequests` periode yang sama dengan matematika CloudWatch metrik. Misalnya, gunakan`FILL(m5xx, 0) / m_executed`, di mana `m5xx` dan `m_executed` apakah ID dari dua metrik. Karena `HTTPResponseStatusClass5XX` jarang, gunakan `FILL` agar periode tanpa kesalahan mengevaluasi ke 0 alih-alih tidak memiliki nilai. `HTTPResponseStatusClass5XX`juga menghitung tanggapan 5xx yang dikembalikan Lambda sebelum permintaan mencapai lingkungan eksekusi. `ExecutedRequests`tidak menghitung tanggapan ini, sehingga rasio dapat melebih-lebihkan tingkat selama gangguan layanan.

Minta metrik di Wilayah asal agregat lalu lintas dari setiap Wilayah yang dilayani titik akhir, sehingga satu alarm di Wilayah asal mencakup seluruh titik akhir. Bangun alarm dan dasbor pada nilai Wilayah

rumah. Jangan mengandalkan metrik ini di Wilayah lain; lihatthe section called “Telemetri di setiap Wilayah”.

Lambda memancarkan metrik permintaan saat permintaan selesai. Untuk permintaan yang berjalan lama, titik data dapat muncul hingga batas waktu permintaan setelah permintaan dimulai, sehingga kesalahan dari permintaan lama dapat mencapai alarm Anda terlambat. Untuk batas waktu permintaan, lihatthe section called “Batas waktu”.

Metrik lingkungan eksekusi

Metrik lingkungan eksekusi menggambarkan sumber daya yang digunakan oleh lingkungan eksekusi yang melayani Fungsi Web Anda. Lambda menerbitkannya di `AWS/Lambda/Web` namespace dengan `FunctionName` dimensi, dan memusatkannya ke Wilayah rumah Fungsi Web.

Lambda memancarkan metrik ini sekali per menit untuk setiap lingkungan eksekusi aktif, sehingga metrik tersebut bisa jarang jika tidak ada lingkungan eksekusi aktif. Nilai bisa 0. Lingkungan eksekusi dapat mempublikasikan lebih dari satu titik data dalam satu menit, misalnya ketika diinisialisasi dan ketika dimatikan, jadi jangan gunakan `SampleCount` statistik untuk menghitung lingkungan aktif. Untuk menghitungnya untuk titik akhir, gunakan`EndpointActiveEnvironments`.

Metrik	Unit	Statistik	Deskripsi
`EnvironmentActiveTime`	`Milliseconds`	`Sum`	Waktu jam dinding di mana lingkungan eksekusi fungsi aktif, termasuk inisialisasi. Setiap titik data adalah waktu aktif satu lingkungan sejak titik data sebelumnya, sehingga Jumlah selama periode adalah total waktu lingkungan yang dikonsumsi.
`EnvironmentCPUTime`	`Milliseconds`	`Sum`	Waktu vCPU yang digunakan oleh lingkungan eksekusi fungsi, termasuk inisialisasi.
`EnvironmentMaxMemoryUsage`	`Megabytes`	`Maximum`	Memori puncak yang telah digunakan lingkungan eksekusi sejauh ini dalam masa hidupnya.
`EnvironmentConcurrentRequests`	`Count`	`Maximum`	Jumlah permintaan tertinggi yang dilayani oleh satu lingkungan eksekusi pada waktu yang sama selama periode tersebut.

Metrik	Unit	Statistik	Deskripsi
`ResponseBytes`	`Bytes`	`Sum`	Jumlah byte respons yang dikirim oleh lingkungan eksekusi fungsi.

Metrik lingkungan eksekusi tidak menyertakan `EndpointName` dimensi. Mereka mencakup semua titik akhir dari Fungsi Web.

Penggunaan regional dan metrik pelambatan

Metrik penggunaan dan pembatasan menunjukkan seberapa dekat akun dan titik akhir Anda dengan batas Regional mereka, dan batas mana yang membatasi permintaan. Kuota tingkat permintaan akun dan kuota vCPU Arm64 akun berlaku secara terpisah di masing-masing, Wilayah AWS sehingga Lambda menerbitkan metrik ini di setiap Wilayah dan tidak memusatkannya. Untuk melihat penggunaan Wilayah, lihat metrik di Wilayah tersebut. Untuk kuota, lihatthe section called “Fungsi Web Lambda”.

Metrik ini dipublikasikan di `AWS/Lambda/Web/Usage` namespace dengan`FunctionName`,`EndpointName`, dan dimensi. `ExecutionRegion` Mereka menggunakan `Count` unit.

Metrik	Statistik	Saat Lambda memancarkannya	Deskripsi
`ExecutedRequests`	`Sum`	Hanya untuk periode di mana setidaknya satu permintaan mencapai lingkungan eksekusi di Wilayah.	Jumlah permintaan yang mencapai lingkungan eksekusi di Wilayah.
`EndpointActiveEnvironments`	`Maximu`	Dengan permintaan ke titik akhir. Nilainya bisa 0. Tidak ada titik data sementara titik akhir tidak menerima permintaan.	Jumlah lingkungan eksekusi aktif dari titik akhir di Wilayah. Bandingkan nilai ini dengan`maxEnvironments` .

Metrik	Statistik	Saat Lambda memancarkannya	Deskripsi
AccountRequestThrottles	Sum	Jarang. Hanya untuk periode dengan setidaknya satu throttle seperti itu.	Jumlah permintaan ke titik akhir yang dibatasi karena akun mencapai kuota tingkat permintaannya di Wilayah.
EndpointRequestThrottles	Sum	Jarang. Hanya untuk periode dengan setidaknya satu throttle seperti itu.	Jumlah permintaan yang dibatasi karena titik akhir mencapai batas tingkat permintaan (). rateLimit
AccountActiveVCPUArm64Throttles	Sum	Jarang. Hanya untuk periode dengan setidaknya satu throttle seperti itu.	Jumlah permintaan ke titik akhir yang dibatasi karena melayani mereka akan melebihi kuota vCPU Arm64 akun di Wilayah. Seperti metrik throttle lainnya, ini hanya menghitung throttle titik akhir ini. Untuk melihat total akun, jumlahkan di seluruh titik akhir.
EndpointActiveEnvironmentThrottles	Sum	Jarang. Hanya untuk periode dengan setidaknya satu throttle seperti itu.	Jumlah permintaan yang dibatasi karena melayani mereka akan melebihi pengaturan titik akhir. maxEnvironments

Lambda mengatribusikan setiap permintaan yang dibatasi ke satu batas, jadi permintaan yang dibatasi akan menambah paling banyak satu metrik throttle. Meminta agar Lambda melambat karena terlalu banyak lingkungan eksekusi yang diinisialisasi pada saat yang sama mengembalikan HTTP 429 dengan jenis Platform.EndpointConcurrentInitThrottle kesalahan tetapi tidak menambah metrik throttle apa pun. Permintaan yang dibatasi juga diperhitungkan HTTPResponseStatusClass4XX dalam namespace. AWS/Lambda/Web

Untuk memantau PerRegion titik akhir MultiRegion atau titik akhir, tampilan atau alarm tentang metrik penggunaan di setiap Wilayah, atau menggabungkannya di dasbor CloudWatch Lintas wilayah.

Menggunakan metrik dengan fitur lain CloudWatch

Karena `AWS/Lambda/Web` metrik terpusat di Wilayah asal, Anda dapat menggunakannya dengan CloudWatch fitur standar di Wilayah tersebut, termasuk:

- Amazon CloudWatch Observability Access Manager — Bagikan metrik Fungsi Web dengan akun pemantauan pusat sehingga satu akun dapat mengamati banyak akun aplikasi.
- Dasbor dan alarm — Bangun dasbor dan alarm di metrik Wilayah asal untuk mencakup seluruh penerapan Multi-wilayah.

Mengatur alarm pada respons 5xx

Buat CloudWatch alarm untuk mendeteksi masalah seperti tingkat kesalahan tinggi atau pelambatan.

Perintah berikut membuat alarm yang memicu ketika penelepon menerima lebih dari lima respons 5xx dari titik akhir dalam periode lima menit.

```
aws cloudwatch put-metric-alarm \
    --alarm-name my-web-app-prod-5xx \
    --namespace AWS/Lambda/Web \
    --metric-name HTTPResponseStatusClass5XX \
    --statistic Sum \
    --period 300 \
    --threshold 5 \
    --comparison-operator GreaterThanThreshold \
    --evaluation-periods 1 \
    --treat-missing-data notBreaching \
    --dimensions Name=FunctionName,Value=my-web-app Name=EndpointName,Value=prod \
    --alarm-actions arn:aws:sns:us-east-1:111122223333:my-alerts
```

Ganti nilai-nilai berikut:

- *my-web-app-prod-5xx*— Sebuah nama untuk alarm.
- *my-web-app*— Nama Fungsi Web Anda.
- *prod*— Nama titik akhir.
- *arn:aws:sns:us-east-1:111122223333:my-alerts*— ARN dari topik Amazon SNS yang menerima pemberitahuan alarm.

`--treat-missing-data notBreaching`penting di sini. `HTTPResponseStatusClass5XX`adalah metrik yang jarang, jadi periode tanpa respons 5xx tidak menghasilkan titik data. Tanpa pengaturan ini, alarm akan memasuki `INSUFFICIENT_DATA` status selama periode tersebut.

> **Note**
>
> Metrik Fungsi Web tidak menyertakan dimensi revisi. Untuk mengatribusikan kesalahan ke revisi selama penerapan canary, tetapkan variabel lingkungan yang mengidentifikasi build, seperti, di setiap revisi`APP_VERSION`, dan sertakan nilainya dalam log terstruktur atau sebagai dimensi dalam metrik yang Anda publikasikan dengan format metrik yang disematkan. Peristiwa log sistem seperti `platform.initStart` juga merekam `revisionId` bahwa setiap lingkungan eksekusi berjalan, dan header `lambda-web-logstream` respons mengidentifikasi aliran log dari lingkungan eksekusi yang melayani permintaan. Untuk prosedur rollback, lihat. the section called “Berguling kembali”

Log Fungsi Web di CloudWatch Log Amazon

AWS Lambda menangkap semua yang ditulis aplikasi Anda ke output standar dan kesalahan standar dan mengirimkannya ke Amazon CloudWatch Logs, bersama dengan log sistem yang dihasilkan Lambda sendiri. DalamNode.js, output dari setiap `console` metode, seperti`console.log`,`console.info`, dan`console.error`, ditangkap secara otomatis dan disaring oleh `applicationLogLevel` pengaturan. Library logging yang menulis ke output standar atau kesalahan standar juga ditangkap. Anda tidak memerlukan agen logging atau Amazon CloudWatch Logs SDK.

Lambda menulis `console` output sebagai peristiwa log JSON dan menambahkan`requestId`,`endpointName`, dan `region` bidang ke setiap acara. Sebagai contoh, lihat the section called “Menulis log terstruktur”.

Lambda mengumpulkan log di masing-masing Wilayah AWS tempat Fungsi Web Anda berjalan dan kemudian memusatkannya ke Wilayah rumah fungsi. `HomeRegion`Titik akhir hanya berjalan di Wilayah asal. `PerRegion`Titik akhir `MultiRegion` atau titik akhir juga berjalan di Wilayah lain yang Anda tentukan, sehingga Lambda memusatkan log dari masing-masing Wilayah tersebut. Untuk informasi selengkapnya, lihat the section called “Jenis titik akhir”.

Grup log

Secara default, Lambda mengirimkan log Fungsi Web Anda ke grup log yang dinamai fungsi:

```
/aws/lambda/web/function-name
```

Bersamaan dengan grup log terpusat, Lambda menggunakan grup log pendamping untuk menyimpan log In-region sebelum disentralisasi. Nama pendamping adalah nama grup log terpusat diikuti oleh`.local`, untuk grup log default dan grup log kustom. Untuk grup log default, pendampingnya adalah:

```
/aws/lambda/web/function-name.local
```

Lambda menulis log ke grup `.local` log di setiap Wilayah tempat permintaan dijalankan, termasuk Wilayah asal, dan kemudian memusatkannya ke dalam grup log tanpa akhiran di Wilayah beranda. Setiap grup `.local` log hanya menyimpan log dari Wilayahnya sendiri. Lambda tidak menghapus peristiwa dari grup `.local` log setelah sentralisasi. Pengaturan retensi grup log menentukan berapa lama mereka tetap ada.

Grup log tidak ada sampai Fungsi Web Anda melayani permintaan pertamanya. Grup log terpusat di Wilayah beranda mungkin muncul setelah grup `.local` log. Untuk waktu sentralisasi, lihatthe section called “Latensi sentralisasi”.

Lambda membuat grup log default dan setiap grup log dengan setelan `.local` CloudWatch Log Amazon default, sehingga peristiwanya tidak akan kedaluwarsa kecuali Anda menetapkan kebijakan penyimpanan. Anda dapat mengubah kebijakan retensi, enkripsi, dan sumber daya setelah Lambda membuatnya. Tetapkan ini pada grup `.local` log di setiap Wilayah tempat titik akhir berjalan dan pada grup log terpusat di Wilayah asal.

Karena grup `.local` log tidak menunggu sentralisasi, ini berguna untuk men-debug masalah di Wilayah tertentu. Grup log terpusat menyimpan log dari setiap Wilayah, jadi buat kueri bersama, filter langganan, filter metrik, dan otomatisasi yang harus mencakup seluruh aplikasi pada grup log terpusat. Untuk bagaimana telemetri berperilaku di setiap Wilayah, lihat. the section called “Telemetri di setiap Wilayah”

Menggunakan grup log kustom

Anda dapat mengirim log Fungsi Web Anda ke grup log yang Anda pilih, bukan default. Gunakan grup log kustom saat Anda menginginkan kebijakan penyimpanan, enkripsi, atau sumber daya

sebelum peristiwa log pertama tiba. Grup log kustom juga memungkinkan Anda mengelompokkan log dari beberapa fungsi bersama-sama. Untuk menerapkan pengaturan tersebut sejak awal, buat grup log dan `.local` pendampingnya sendiri sebelum Anda mengarahkan lalu lintas ke revisi. Jika salah satu grup log tidak ada, Lambda membuatnya pada penggunaan pertama dengan pengaturan default, dan Anda dapat memperbarui pengaturannya sesudahnya. Lambda dapat membuat grup log hanya jika peran eksekusi memiliki izin untuk membuat grup log. Untuk izin yang diperlukan, lihat the section called “Peran eksekusi”.

Untuk mengirim log ke grup log kustom (AWS CLI)

1. Di Wilayah beranda, buat grup log dan `.local` pendampingnya, dan atur retensi yang Anda inginkan pada masing-masing. Untuk `PerRegion` titik akhir `MultiRegion` atau titik akhir, buat juga grup `.local` log di setiap Wilayah lain yang dilayani titik akhir, dengan menjalankan `.local` perintah yang `--region` disetel ke Wilayah tersebut.

```
aws logs create-log-group --log-group-name /my-app/web-logs
aws logs create-log-group --log-group-name /my-app/web-logs.local

aws logs put-retention-policy \
    --log-group-name /my-app/web-logs \
    --retention-in-days 30

aws logs put-retention-policy \
    --log-group-name /my-app/web-logs.local \
    --retention-in-days 30
```

2. Buat revisi yang menentukan grup log di `telemetryConfig` bagian. `serviceConfig`

```
aws lambda-web create-web-function-revision \
    --function-name my-web-app \
    --build-config '{
      "codeConfig": {
        "s3Object": { "bucket": "my-bucket", "key": "my-app.zip" }
      },
      "runtimeConfig": { "runtime": "nodejs24.x" }
    }' \
    --service-config '{
      "executionRoleArn": "arn:aws:iam::111122223333:role/my-web-function-role",
      "telemetryConfig": {
        "loggingConfig": {
          "logGroup": "/my-app/web-logs"
```

```
        }
      }
    }'
```

3. Rute lalu lintas ke revisi baru. Untuk informasi selengkapnya, lihat the section called “Pola deployment”.

Karena konfigurasi logging adalah bagian dari revisi, itu tidak dapat diubah untuk revisi itu. Untuk mengubah grup log nanti, buat revisi lain. Untuk informasi selengkapnya, lihat the section called “Revisi”.

Kontrol logging

`loggingConfig`Bagian ini mengontrol tujuan dan tingkat keparahan minimum log. Pengaturan ini adalah bagian dari revisi.

Pengaturan	Nilai	Default	Deskripsi
`logGroup`	Nama grup log	`/aws/lambda/web/`*`function-name`*	Grup CloudWatch log Amazon Logs yang menerima log Anda.
`applicationLogLevel`	`TRACE` | `DEBUG` | `INFO` | `WARN` | `ERROR` | `FATAL`	`INFO`	Tingkat keparahan minimum log aplikasi Anda yang dikirim Lambda ke Amazon CloudWatch Logs. Log di bawah level ini dijatuhkan. Node.jstidak memiliki metode `FATAL` konsol, jadi tingkat `FATAL` berperilaku sebagai`ERROR`.
`systemLogLevel`	`DEBUG` | `INFO` | `WARN`	`INFO`	Tingkat keparahan minimum log Lambda-generated sistem yang dikirim Lambda ke Amazon CloudWatch Logs. Untuk tingkat setiap peristiwa sistem, lihatthe section called “Log sistem”.

Contoh berikut membuat revisi yang menjatuhkan log aplikasi di bawah iniWARN:

```
aws lambda-web create-web-function-revision \
    --function-name my-web-app \
    --build-config '{
      "codeConfig": {
        "s3Object": { "bucket": "my-bucket", "key": "my-app.zip" }
      },
      "runtimeConfig": { "runtime": "nodejs24.x" }
    }' \
    --service-config '{
      "executionRoleArn": "arn:aws:iam::111122223333:role/my-web-function-role",
      "telemetryConfig": {
        "loggingConfig": {
          "applicationLogLevel": "WARN",
          "systemLogLevel": "INFO"
        }
      }
    }'
```

`applicationLogLevel`Peningkatan produksi mengurangi volume log dan biaya. Turunkan sementara ke DEBUG atau TRACE ketika Anda sedang menyelidiki suatu masalah.

Log sistem

Selain output aplikasi Anda, grup log Anda menerima log sistem yang dihasilkan Lambda tentang siklus hidup lingkungan eksekusi, seperti inisialisasi lingkungan dan shutdown. Gunakan `systemLogLevel` untuk mengontrol berapa banyak yang dipancarkan Lambda ini.

Log sistem berguna untuk menghubungkan perilaku aplikasi dengan peristiwa platform. Misalnya, lonjakan latensi yang sejalan dengan entri inisialisasi lingkungan menunjukkan awal yang dingin daripada kode aplikasi Anda. Untuk siklus hidup yang dijelaskan oleh log ini, lihat. the section called “Siklus hidup lingkungan”

Fungsi Web memancarkan peristiwa sistem berikut. Atur `systemLogLevel` ke level terendah yang Anda butuhkan. Misalnya, atur DEBUG ke menerima`platform.requestComplete`, atau hanya menerima kegagalan dan peringatan dropped-log. WARN

Tipe peristiwa	Tingkat	Catatan
`platform.initStart`	`INFO`	Tidak ada
`platform.initReport`	`INFO`pada kesuksesan, `WARN` pada kegagalan	Kegagalan menunjukkan kesalahan inisialisasi, seperti kesalahan saat mengimpor kode Anda.
`platform.invokeStart`	`INFO`	Tidak ada
`platform.requestComplete`	`DEBUG`	Dipancarkan sekali untuk setiap permintaan HTTP, termasuk permintaan yang gagal. Karena volumenya tinggi, tidak dipancarkan pada tingkat log sistem default.
`platform.invokeSummary`	`INFO`	Berkala. Dipancarkan setiap 60 detik untuk setiap lingkungan eksekusi aktif.
`platform.invokeReport`	`INFO`pada kesuksesan, `WARN` pada kegagalan	
`platform.spinDownStart`	`INFO`	
`platform.spinDownReport`	`INFO`pada kesuksesan, `WARN` pada kegagalan	Kegagalan menunjukkan bahwa aplikasi Anda tidak keluar dalam waktu 1 detik setelah menerima `SIGTERM` (`Runtime.SpinDownTimeout`) atau keluar dengan error (`Runtime.ExitError`).
`platform.logsDropped`	`WARN`	Menunjukkan bahwa Lambda menghapus peristiwa log.

Untuk bidang di setiap acara sistem, lihatthe section called “Skema log sistem”.

Header `lambda-web-request-id` respons mengidentifikasi permintaan, dan header `lambda-web-logstream` respons mengidentifikasi aliran log untuk lingkungan yang menanganinya. Gunakan nilai-nilai ini untuk mengkorelasikan respons klien dengan log aplikasi dan sistem. Untuk informasi selengkapnya, lihat the section called “Header respons”.

Untuk menemukan peristiwa dari aliran log yang diidentifikasi dalam grup log terpusat di Wilayah beranda, filter berdasarkan nama aliran sebagai awalan. Anda juga dapat membaca aliran dengan nama persisnya dari grup `.local` log di Wilayah yang melayani permintaan.

```
aws logs filter-log-events \
    --log-group-name /aws/lambda/web/my-web-app \
    --log-stream-name-prefix "LAMBDA_WEB_LOGSTREAM" \
    --region us-east-1
```

Untuk mengikuti log terbaru saat mereproduksi masalah, gunakan:

```
aws logs tail /aws/lambda/web/my-web-app \
    --follow \
    --region us-east-1
```

Menulis log terstruktur

Berikan objek ke `console` metode sehingga Lambda menulisnya sebagai objek JSON di lapangan. `message` Anda kemudian dapat menanyakan setiap bidang di Amazon CloudWatch Logs Insights daripada mencocokkan substring. Jangan melewatkan string yang Anda buat dengan`JSON.stringify`. Lambda menulis string sebagai nilai string yang lolos, sehingga bidangnya tidak dapat ditanyakan.

```
function logRequest(req, statusCode, durationMs) {
  console.info({
    method: req.method,
    path: req.url,
    statusCode: statusCode,
    durationMs: durationMs
  });
}
```

Lambda menulis peristiwa log yang mirip dengan yang berikut ini:

```
{
```

```
  "timestamp": "2026-09-25T11:29:02.711Z",
  "level": "INFO",
  "message": {
    "method": "GET",
    "path": "/hello-world",
    "statusCode": 200,
    "durationMs": 100
  },
  "requestId": "US3NHLM77IRXDFLBADYHGIZZNM",
  "endpointName": "prod",
  "region": "us-east-1"
}
```

Anda kemudian dapat menanyakan bidang secara langsung. Misalnya, kueri Amazon CloudWatch Logs Insights berikut menampilkan 20 permintaan terbaru yang menampilkan status 5xx:

```
fields @timestamp, message.method, message.path, message.statusCode, message.durationMs
| filter message.statusCode >= 500
| sort @timestamp desc
| limit 20
```

Untuk memublikasikan metrik kustom dari log Anda, bukan hanya menanyainya, gunakan format metrik yang disematkan. Untuk informasi selengkapnya, lihat the section called “Metrik-metrik kustom”.

Menggunakan fitur Amazon CloudWatch Logs dengan log Anda

Karena log dipusatkan ke Wilayah asal, Anda dapat menggunakan fitur CloudWatch Log Amazon standar di Wilayah tersebut, termasuk:

- Live Tail - Streaming peristiwa log saat mereka tiba saat Anda mereproduksi masalah. Peristiwa mencapai grup log terpusat setelah sentralisasi, jadi untuk penundaan terpendek, jalankan Live Tail pada grup `.local` log di Wilayah yang melayani permintaan Anda.
- Wawasan Log - Bidang kueri dalam peristiwa log JSON Anda.
- Sentralisasi log — Teruskan log Anda ke akun atau tujuan lain menggunakan fitur sentralisasi CloudWatch Log Amazon, jika Anda menggabungkan log di luar akun fungsi.
- Filter langganan dan filter metrik — Memicu pemrosesan hilir atau dapatkan metrik dari pola log.

Terapkan fitur yang harus mencakup seluruh aplikasi ke grup log terpusat di Wilayah asal. Grup `.local` log hanya menyimpan log dari Wilayahnya sendiri. Lihat informasi yang lebih lengkap di the section called “Telemetri di setiap Wilayah”.

Skema peristiwa log sistem untuk Fungsi Web Lambda

Lambda menulis peristiwa sistem berikut ke grup log Fungsi Web Anda. Setiap peristiwa adalah objek JSON dengan bidang amplop yang dijelaskan dalam the section called “Amplop acara” dan `record` objek yang bidangnya bergantung pada jenis acara. Tingkat setiap acara menentukan apakah Lambda mengirimkannya. Untuk informasi selengkapnya, lihat the section called “Log sistem”.

Tipe peristiwa	Rekam objek	Deskripsi
`platform.initStart`	`PlatformInitStart`	Inisialisasi lingkungan dimulai.
`platform.initReport`	`PlatformInitReport`	Laporan inisialisasi lingkungan, apakah berhasil atau gagal.
`platform.invokeStart`	`PlatformInvokeStart`	Lingkungan mulai melayani permintaan.
`platform.requestComplete`	`PlatformRequestComplete`	Satu permintaan HTTP selesai. Masuk di DEBUG saja.
`platform.invokeSummary`	`PlatformInvokeSummary`	Ringkasan aktivitas lingkungan, dipancarkan setiap 60 detik.
`platform.invokeReport`	`PlatformInvokeReport`	Laporan seluruh periode penyajian lingkungan, dipancarkan saat berhenti melayani.
`platform.spinDownStart`	`PlatformSpinDownStart`	Shutdown lingkungan dimulai.
`platform.spinDownReport`	`PlatformSpinDownReport`	Laporan penutupan lingkungan.

Tipe peristiwa	Rekam objek	Deskripsi
`platform.logsDropped`	`PlatformLogsDropped`	Catatan log dijatuhkan karena log diproduksi lebih cepat daripada yang dapat dikirimkan.

Dalam daftar bidang di halaman ini, tanda tanya (?) setelah nama bidang berarti bahwa bidang tersebut mungkin tidak ada. Jenis yang bukan tipe JSON, seperti`InitType`,`Status`, atau`InvokeSummaryMetrics`, didefinisikan dalamthe section called “Jenis objek bersama”, dan masing-masing terhubung ke definisinya.

`Platform.initStart`

Fase inisialisasi lingkungan dimulai.

Tingkat: `INFO`

`record`Objek memiliki bidang berikut:

- `functionName`(`String`) — Nama fungsi.
- `revisionId`(`String`) — Revisi yang dijalankan lingkungan.
- `instanceId`(`String`) — Pengidentifikasi lingkungan eksekusi.
- `initializationType`(`InitType`) — Bagaimana Lambda menginisialisasi lingkungan.
- `runtimeVersion`(`String`) — Versi runtime yang dijalankan lingkungan.
- `runtimeVersionArn`(`String`) — Nama Sumber Daya Amazon (ARN) dari versi runtime.

```
{
  "timestamp": "2026-09-25T11:12:30.893Z",
  "level": "INFO",
  "type": "platform.initStart",
  "region": "us-east-1",
  "record": {
    "runtimeVersion": "nodejs24.x:nodejs.web24.v51",
    "runtimeVersionArn": "arn:aws:lambda:us-east-1::runtime/nodejs24.x/version/
nodejs.web24.v51",
    "functionName": "my-web-app",
    "revisionId": "rev-fs0fq",
    "instanceId": "52754708-44f1-407a-a1fd-6d2a98175a45",
    "initializationType": "native-http"
```

```
  }
}
```

`platform.initReport`

Laporan keseluruhan fase inisialisasi.

Level: `INFO` ketika status sukses, `WARN` ketika kegagalan.

`record`Objek memiliki bidang berikut:

- `status`(`Status`) — Hasil dari fase inisialisasi.
- `errorType?`(`String`) - Hadir saat status gagal. Nilai-nilai sepertiRuntime.ImportCodeError,Application.InitCodeError, danRuntime.InitTimeout.
- `initializationType`(`InitType`) — Bagaimana Lambda menginisialisasi lingkungan.
- `metrics`(`InitReportMetrics`) — Metrik untuk fase inisialisasi.

Contoh sukses:

```
{
  "timestamp": "2026-09-25T11:12:31.143Z",
  "level": "INFO",
  "type": "platform.initReport",
  "region": "us-east-1",
  "record": {
    "status": "success",
    "initializationType": "native-http",
    "metrics": {
      "totalDurationMs": 304.000,
      "totalCpuTimeMs": 143.000,
      "maxMemoryUsageMb": 142
    }
  }
}
```

Contoh kegagalan, di mana modul handler gagal memuat:

```
{
```

```
  "timestamp": "2026-09-25T11:16:04.164Z",
  "level": "WARN",
  "type": "platform.initReport",
  "region": "us-east-1",
  "record": {
    "status": "failure",
    "errorType": "Runtime.ImportCodeError",
    "initializationType": "native-http",
    "metrics": {
      "totalDurationMs": 320.000,
      "totalCpuTimeMs": 137.000,
      "maxMemoryUsageMb": 138
    }
  }
}
```

`platform.invokeStart`

Lingkungan selesai menginisialisasi dan mulai melayani permintaan. Lambda memancarkan peristiwa ini sekali per lingkungan, bukan sekali per permintaan, dan hanya ketika inisialisasi berhasil.

Tingkat: `INFO`

`record`Objek memiliki bidang berikut:

- `functionName`(`String`) — Nama fungsi.
- `revisionId`(`String`) — Revisi yang dijalankan lingkungan.
- `instanceId`(`String`) — Pengidentifikasi lingkungan eksekusi.

```
{
  "timestamp": "2026-09-25T11:12:31.144Z",
  "level": "INFO",
  "type": "platform.invokeStart",
  "region": "us-east-1",
  "record": {
    "functionName": "my-web-app",
    "revisionId": "rev-fs0fq",
    "instanceId": "52754708-44f1-407a-a1fd-6d2a98175a45"
  }
}
```

`platform.requestComplete`

Satu permintaan HTTP selesai, berhasil atau tidak.

Level:`DEBUG`, jadi Lambda mengirimkannya hanya saat itu. `systemLogLevel` `DEBUG`

`record`Objek memiliki bidang berikut:

- `requestId`(`String`) — ID permintaan.
- `endpointName`(`String`) — Titik akhir yang melayani permintaan.
- `result`(`Integer`) - Kode status HTTP dikembalikan ke klien.
- `errorType?`(`String`) — Hadir saat Lambda gagal dalam permintaan. Nilai-nilai seperti `Runtime.RequestTimeout` dan`Platform.InvalidRequest`.
- `metrics`(`RequestCompleteMetrics`) — Metrik untuk permintaan.

Contoh sukses:

```
{
  "timestamp": "2026-09-25T11:12:31.154Z",
  "level": "DEBUG",
  "type": "platform.requestComplete",
  "region": "us-east-1",
  "record": {
    "requestId": "KXH25FBDZ33GP4TQ4ARAUYSTSA",
    "endpointName": "default",
    "result": 200,
    "metrics": {
      "durationMs": 4.484,
      "responseBytes": 40
    }
  }
}
```

Contoh batas waktu:

```
{
  "timestamp": "2026-09-25T11:12:34.926Z",
  "level": "DEBUG",
  "type": "platform.requestComplete",
```

```
  "region": "us-east-1",
  "record": {
    "requestId": "5SXHGVX2GD64ND6NQVEBAY235Q",
    "endpointName": "default",
    "result": 504,
    "errorType": "Runtime.RequestTimeout",
    "metrics": {
      "durationMs": 3001.229,
      "responseBytes": 0
    }
  }
}
```

`platform.invokesummary`

Merangkum satu jendela sekitar 60 detik aktivitas lingkungan. Lambda memancarkannya untuk setiap lingkungan hidup, termasuk jendela siaga di mana 0. `requests`

Tingkat: `INFO`

`record`Objek memiliki bidang berikut:

- `startTime`(`String`) — Awal jendela ringkasan (RFC 3339).
- `instanceId`(`String`) — Pengidentifikasi lingkungan eksekusi.
- `metrics`(`InvokeSummaryMetrics`) — Metrik untuk jendela.

```
{
  "timestamp": "2026-09-25T11:13:31.143Z",
  "level": "INFO",
  "type": "platform.invokeSummary",
  "region": "us-east-1",
  "record": {
    "startTime": "2026-09-25T11:12:31.144547426Z",
    "instanceId": "52754708-44f1-407a-a1fd-6d2a98175a45",
    "metrics": {
      "requests": 4,
      "concurrentRequests": 1,
      "cpuTimeMs": 420.000,
      "maxMemoryUsageMb": 154
    }
  }
```

```
}
```

`platform.InvokeReport`

Laporan keseluruhan periode pelayanan lingkungan, dari InvokeStart hingga berhenti melayani. Lambda memancarkannya sekali per lingkungan, saat shutdown.

Level: `INFO` ketika status sukses, `WARN` ketika kegagalan.

`record`Objek memiliki bidang berikut:

- `status`(`Status`) — Hasil dari periode melayani.
- `errorType?`(`String`) - Hadir saat status gagal. Nilai-nilai sepertiRuntime.Exit,Runtime.ExitError, danRuntime.OutOfMemory.
- `instanceId`(`String`) — Pengidentifikasi lingkungan eksekusi.
- `metrics`(`InvokeReportMetrics`) — Metrik untuk periode penyajian.

Batas waktu permintaan tidak gagal dalam periode penyajian. `status`Tinggal`success`.

Contoh sukses:

```
{
  "timestamp": "2026-09-25T11:19:11.858Z",
  "level": "INFO",
  "type": "platform.invokeReport",
  "region": "us-east-1",
  "record": {
    "status": "success",
    "instanceId": "52754708-44f1-407a-a1fd-6d2a98175a45",
    "metrics": {
      "totalCpuTimeMs": 973.000,
      "totalDurationMs": 400723.000,
      "totalRequests": 4,
      "concurrentRequests": 1,
      "maxMemoryUsageMb": 157
    }
  }
}
```

Contoh kegagalan:

```
{
  "timestamp": "2026-09-25T11:19:11.858Z",
  "level": "WARN",
  "type": "platform.invokeReport",
  "region": "us-east-1",
  "record": {
    "status": "failure",
    "errorType": "Runtime.OutOfMemory",
    "instanceId": "52754708-44f1-407a-a1fd-6d2a98175a45",
    "metrics": {
      "totalCpuTimeMs": 973.000,
      "totalDurationMs": 400723.000,
      "totalRequests": 4,
      "concurrentRequests": 1,
      "maxMemoryUsageMb": 512
    }
  }
}
```

`platform.spin DownStart`

Shutdown lingkungan dimulai.

Tingkat: `INFO`

`record`Objek memiliki bidang berikut:

- `functionName`(`String`) — Nama fungsi.
- `revisionId`(`String`) — Revisi yang dijalankan lingkungan.
- `instanceId`(`String`) — Pengidentifikasi lingkungan eksekusi.

```
{
  "timestamp": "2026-09-25T11:19:11.858Z",
  "level": "INFO",
  "type": "platform.spinDownStart",
  "region": "us-east-1",
  "record": {
    "functionName": "my-web-app",
    "revisionId": "rev-fs0fq",
    "instanceId": "52754708-44f1-407a-a1fd-6d2a98175a45"
  }
```

```
}
```

`platform.spin DownReport`

Laporan keseluruhan dari shutdown.

Lambda mengirim `SIGTERM` ke aplikasi Anda dan menunggu hingga 1 detik untuk keluar. Jika aplikasi masih berjalan, Lambda menghentikannya `SIGKILL` dan melaporkan kegagalan dengan jenis kesalahan. `Runtime.SpinDownTimeout`

Level: `INFO` ketika status sukses, `WARN` ketika kegagalan.

`record`Objek memiliki bidang berikut:

- `status`(`Status`) — Hasil dari fase shutdown.
- `errorType?`(`String`) - Hadir saat status gagal. Nilai-nilai seperti `Runtime.SpinDownTimeout` dan`Runtime.ExitError`.
- `metrics`(`SpinDownReportMetrics`) — Metrik untuk fase shutdown.

Contoh sukses:

```
{
  "timestamp": "2026-09-25T11:19:11.879Z",
  "level": "INFO",
  "type": "platform.spinDownReport",
  "region": "us-east-1",
  "record": {
    "status": "success",
    "metrics": {
      "totalDurationMs": 21.000,
      "totalCpuTimeMs": 8.000,
      "maxMemoryUsageMb": 157
    }
  }
}
```

Contoh kegagalan:

```
{
  "timestamp": "2026-09-25T11:19:12.879Z",
```

```
  "level": "WARN",
  "type": "platform.spinDownReport",
  "region": "us-east-1",
  "record": {
    "status": "failure",
    "errorType": "Runtime.SpinDownTimeout",
    "metrics": {
      "totalDurationMs": 1021.000,
      "totalCpuTimeMs": 8.000,
      "maxMemoryUsageMb": 157
    }
  }
}
```

`platform.logsDropped`

Catatan log dijatuhkan karena log diproduksi lebih cepat daripada yang dapat dikirimkan.

Tingkat: `WARN`

`record`Objek memiliki bidang berikut:

- `reason`(`String`) — Mengapa catatan dijatuhkan.
- `droppedRecords`(`Integer`) — Catatan turun sejak rekor terakhir yang dikirimkan.
- `droppedBytes`(`Integer`) — Bytes turun sejak rekor terakhir yang dikirimkan.

Kedua hitungan terakumulasi di seluruh tetes berturut-turut dan diatur ulang ke 0 setelah catatan dikirimkan.

```
{
  "timestamp": "2026-09-25T11:20:00.000Z",
  "level": "WARN",
  "type": "platform.logsDropped",
  "region": "us-east-1",
  "record": {
    "reason": "Some logs were dropped because the downstream consumer is slower than
 the logs production rate",
    "droppedRecords": 120,
    "droppedBytes": 48000
  }
}
```

Jenis objek bersama

Jenis berikut muncul di daftar bidang peristiwa di halaman ini.

Amplop acara

Setiap peristiwa sistem memiliki bidang amplop berikut:

- `timestamp`(`String`) — RFC 3339 di UTC.
- `level`(`Level`) — Tingkat acara.
- `type`(`String`) — Jenis acara (`platform.*`).
- `region`(`String`) — Wilayah tempat lingkungan eksekusi berjalan.
- `record`(`Object`) — Bidang khusus acara.

`InitType`

String enum yang menjelaskan bagaimana Lambda menginisialisasi lingkungan. Fungsi Web selalu digunakan`native-http`.

`Tingkat`

String enum untuk tingkat peristiwa sistem:`DEBUG`,`INFO`, atau`WARN`. Peristiwa sistem tidak pernah `ERROR` atau`FATAL`. Laporan peristiwa adalah `INFO` ketika status sukses dan `WARN` ketika itu gagal.

`Status`

String enum yang menggambarkan hasil dari suatu fase: `success` atau`failure`. Ketika itu`failure`, catatan juga berisi`errorType`.

`InitReportMetrics`

- `totalDurationMs`(`Double`) — Wall-clock waktu fase inisialisasi.
- `totalCpuTimeMs`(`Double`) — Waktu CPU yang dikonsumsi selama inisialisasi.
- `maxMemoryUsageMb`(`Integer`) — Memori puncak lingkungan, di MiB.

`RequestCompleteMetrics`

- `durationMs`(`Double`) — Wall-clock waktu permintaan.

- `responseBytes(Integer)` — Byte tubuh respons dikirim.

`InvokeSummaryMetrics`

- `requests(Integer)` — Permintaan selesai di jendela.
- `concurrentRequests(Integer)` — Permintaan bersamaan maksimum di jendela.
- `cpuTimeMs(Double)` — Waktu CPU dikonsumsi di jendela.
- `maxMemoryUsageMb(Integer)` — Memori puncak lingkungan, di MiB.

`InvokeReportMetrics`

- `totalCpuTimeMs(Double)` — Waktu CPU selama periode penyajian.
- `totalDurationMs(Double)` — Wall-clock waktu dari InvokeStart hingga lingkungan berhenti melayani.
- `totalRequests(Integer)` — Total permintaan yang dilayani.
- `concurrentRequests(Integer)` — Permintaan bersamaan maksimum selama periode tersebut.
- `maxMemoryUsageMb(Integer)` — Memori puncak lingkungan, di MiB.

`SpinDownReportMetrics`

- `totalDurationMs(Double)` — Wall-clock waktu fase shutdown.
- `totalCpuTimeMs(Double)` — Waktu CPU yang dikonsumsi selama shutdown.
- `maxMemoryUsageMb(Integer)` — Memori puncak lingkungan, di MiB.

Menerbitkan metrik kustom dengan format metrik yang disematkan

Dengan format metrik CloudWatch tertanam Amazon (EMF), Anda dapat mempublikasikan metrik Anda sendiri dari Fungsi AWS Lambda Web, seperti pesanan yang ditempatkan, tingkat klik cache, atau latensi pembayaran. Aplikasi Anda menulis objek JSON terstruktur ke lognya, dan CloudWatch mengekstrak metrik darinya. Gunakan metrik khusus untuk melacak apa yang dilakukan aplikasi Anda. Metrik yang diterbitkan Lambda hanya menjelaskan hasil permintaan dan penggunaan sumber daya.

Anda juga dapat mempublikasikan metrik kustom dengan memanggil CloudWatch `PutMetricData` API secara langsung, tetapi kami merekomendasikan EMF untuk Fungsi Web:

- Tidak ada latensi permintaan tambahan - Aplikasi Anda tidak memanggil CloudWatch API di jalur yang melayani respons HTTP Anda.
- Tidak ada izin tambahan — Anda tidak memerlukan `cloudwatch:PutMetricData` izin. EMF menggunakan izin Amazon CloudWatch Logs yang sama dengan peran eksekusi yang sudah dibutuhkan untuk log.
- Sentralisasi dengan log Anda — Karena metrik EMF berjalan dengan log Anda, Wilayah asal memiliki nilai dari setiap Wilayah yang dilayani titik akhir. Metrik yang Anda publikasikan hanya `PutMetricData` muncul di Wilayah tempat Anda memanggilnya.

Bagaimana EMF bekerja dengan Fungsi Web

Metrik EMF berjalan dari aplikasi Anda ke CloudWatch melalui langkah-langkah berikut:

1. Aplikasi Anda menulis objek EMF-formatted JSON, diikuti oleh baris baru, langsung ke output standar.
2. Lambda menangkapnya sebagai peristiwa log di Wilayah tempat permintaan dijalankan dan memusatkannya ke beranda Wilayah AWS, sama seperti keluaran log lainnya. Lihat perinciannya di the section called “Beberapa catatan”.
3. CloudWatch mengekstrak metrik dari objek di Wilayah tempat permintaan dijalankan, dan Lambda mereplikasi metrik ke Wilayah asal. Metrik muncul di namespace yang Anda tentukan di objek. Di Wilayah asal, metrik mencakup nilai dari setiap Wilayah yang dilayani titik akhir.

Metrik kustom Anda muncul di namespace Anda sendiri, bukan di. `AWS/Lambda/Web` Namespace itu dicadangkan untuk metrik yang diterbitkan Lambda.

Menerbitkan metrik khusus

Node.jsContoh berikut menerbitkan `CheckoutLatency` metrik dan `OrdersPlaced` metrik di bawah `MyWebApp` namespace, dengan dimensi. `PaymentProvider` Ini menulis objek EMF dengan`process.stdout.write`. Jangan meneruskan objek EMF ke `console` metode. Lambda membungkus `console` output dalam peristiwa log JSON-nya sendiri, dan CloudWatch tidak mengekstrak metrik dari objek yang dibungkus.

```
function publishCheckoutMetrics(provider, latencyMs, orders) {
  process.stdout.write(JSON.stringify({
    _aws: {
      Timestamp: Date.now(),
```

```
      CloudWatchMetrics: [
        {
          Namespace: 'MyWebApp',
          Dimensions: [['PaymentProvider']],
          Metrics: [
            { Name: 'CheckoutLatency', Unit: 'Milliseconds' },
            { Name: 'OrdersPlaced', Unit: 'Count' }
          ]
        }
      ]
    },
    PaymentProvider: provider,
    CheckoutLatency: latencyMs,
    OrdersPlaced: orders,
    // Fields outside the metric definitions are kept as searchable log context.
    requestPath: '/checkout'
  }) + '\n');
}
```

Alih-alih membuat objek EMF sendiri, Anda dapat menggunakan pustaka klien EMF, seperti aws-embedded-metrics untuk di situs web. Node.js GitHub Untuk daftar pustaka, lihat Membuat log dalam format metrik tertanam menggunakan pustaka klien di CloudWatch Panduan Pengguna Amazon. Sebelum Anda mengandalkan pustaka, konfirmasikan bahwa ia menulis objek EMF langsung ke output standar dalam Fungsi Web, bukan melalui `console` atau ke CloudWatch agen.

Perhatikan tiga peran yang dimainkan bidang dalam objek EMF:

- Bidang yang dinamai `Metrics` menjadi CloudWatch metrik.
- Bidang yang dinamai `Dimensions` menjadi dimensi metrik. Jaga agar jumlah nilai dimensi yang berbeda tetap rendah. Setiap kombinasi menciptakan metrik terpisah.
- Bidang lain disimpan dalam peristiwa log sebagai konteks yang dapat Anda kueri dengan Amazon CloudWatch Logs Insights, tetapi tidak membuat metrik.

Gunakan properti terakhir dengan sengaja: lampirkan nilai kardinalitas tinggi seperti ID permintaan atau ID pelanggan sebagai bidang biasa dan bukan sebagai dimensi, sehingga Anda dapat menemukan permintaan spesifik di Wawasan Log tanpa mengalikan jumlah metrik Anda.

Penagihan untuk metrik kustom

EMF menghasilkan konsumsi log dan biaya metrik khusus. Peristiwa log ditagih dengan harga Amazon CloudWatch Logs, dan metrik yang diekstraksi ditagih sebagai metrik khusus. CloudWatch Sentralisasi pertama dari peristiwa log gratis. CloudWatch mengekstrak metrik EMF di Wilayah tempat permintaan dijalankan, dan Lambda mereplikasinya ke Wilayah asal, sehingga permintaan yang disajikan di luar Wilayah asal menghasilkan metrik khusus di kedua Wilayah.

Volume log, jumlah metrik, dan jumlah kombinasi dimensi yang berbeda memengaruhi biaya. Publikasikan metrik pada perincian yang akan Anda alarm atau grafik, dan hindari dimensi kardinalitas tinggi.

Untuk harga saat ini, lihat CloudWatchharga Amazon.

Logging panggilan API Fungsi Web menggunakan AWS CloudTrail

Dengan AWS CloudTrail, Anda dapat melacak tindakan yang diambil oleh pengguna, peran, atau Layanan AWS dalam Fungsi AWS Lambda Web. CloudTrail memberikan catatan tindakan ini. CloudTrail menangkap pemanggilan Fungsi Web sebagai peristiwa data. Anda dapat menggunakan informasi ini untuk menentukan siapa yang memanggil Fungsi Web, kapan, dan dari mana, yang membantu analisis keamanan dan audit operasional.

Peristiwa data Fungsi Web di CloudTrail

Peristiwa data memberikan informasi tentang operasi yang dilakukan pada sumber daya. Misalnya, memanggil titik akhir Fungsi Web adalah peristiwa data. Operasi ini juga dikenal sebagai operasi data-plane, dan seringkali bervolume tinggi. Secara default, CloudTrail tidak mencatat peristiwa data, dan riwayat CloudTrail Peristiwa tidak merekamnya.

Biaya tambahan berlaku untuk peristiwa data. Untuk informasi selengkapnya, lihat harga CloudTrail .

Anda dapat mencatat peristiwa data untuk jenis `AWS::Lambda::WebFunction` sumber daya menggunakan CloudTrail konsol AWS CLI, atau operasi CloudTrail API. CloudTrail mencatat setiap pemanggilan dalam Wilayah AWS yang melayani permintaan, dan `awsRegion` mengidentifikasi Wilayah itu. Tidak seperti log dan metrik, acara pemanggilan tidak terpusat di Wilayah asal. Untuk `PerRegion` titik akhir `MultiRegion` atau titik akhir, gunakan jejak Multi-wilayah atau penyimpanan data peristiwa sehingga pemanggilan di setiap Wilayah yang dilayani titik akhir dicatat.

Tabel berikut menunjukkan tipe peristiwa data untuk Web Functions.

Jenis peristiwa data (konsol)	nilai resources.type	API data masuk CloudTrail
Fungsi Web Lambda	`AWS::Lambda::WebFunction`	`InvokeWebFunctionEndpoint`

CloudTrail mencatat pengidentifikasi kunci berikut untuk pemanggilan Fungsi Web:

Pengidentifikasi	Nilai
`eventSource`	`lambda.amazonaws.com`
`eventName`	`InvokeWebFunctionEndpoint` . Ini adalah nilai untuk memfilter.
Tindakan IAM	`lambda:InvokeWebFunctionEndpoint`
`readOnly`	`false`, karena pemanggilan menjalankan kode fungsi Anda.
`resources.type`	`AWS::Lambda::WebFunction`
`resources.ARN`	Fungsi ARN, bukan titik akhir ARN. Oleh karena itu, pemilih jejak bekerja pada tingkat fungsi.

Pemilih peristiwa lanjutan berikut mencatat pemanggilan untuk satu Fungsi Web saja.

```
[
    {
        "Name": "web-function-invokes",
        "FieldSelectors": [
            { "Field": "eventCategory", "Equals": ["Data"] },
            { "Field": "resources.type", "Equals": ["AWS::Lambda::WebFunction"] },
            { "Field": "resources.ARN", "Equals": ["arn:aws:lambda:us-
east-1:111122223333:web-function/hello-world"] }
        ]
    }
]
```

Untuk mengaktifkan peristiwa data pada jejak, gunakan prosedur berikut.

Untuk mengaktifkan dan memverifikasi peristiwa data Fungsi Web (AWS CLI)

1. Buat atau pilih jejak yang akan merekam peristiwa data. Untuk petunjuknya, lihat Mencatat peristiwa data di Panduan AWS CloudTrail Pengguna.
2. Simpan pemilih acara lanjutan berikut sebagai`web-function-data-events.json`. Ini merekam pemanggilan untuk satu Fungsi Web. Untuk merekam pemanggilan untuk setiap Fungsi Web di akun, hapus pemilih `resources.ARN` bidang.

```
[
  {
    "Name": "web-function-invocations",
    "FieldSelectors": [
      {
        "Field": "eventCategory",
        "Equals": ["Data"]
      },
      {
        "Field": "resources.type",
        "Equals": ["AWS::Lambda::WebFunction"]
      },
      {
        "Field": "resources.ARN",
        "Equals": [
          "arn:aws:lambda:us-east-1:111122223333:web-function/my-web-app"
        ]
      }
    ]
  }
]
```

3. Terapkan pemilih ke jejak:

```
aws cloudtrail put-event-selectors \
    --trail-name my-trail \
    --advanced-event-selectors file://web-function-data-events.json
```

> **Important**
>
> `put-event-selectors`menggantikan penyeleksi acara jejak yang ada. Jika jejak sudah merekam peristiwa lain, sertakan penyeleksi mereka dalam file yang sama.

4. Panggil titik akhir dengan permintaan pengujian.
5. Tanyakan tujuan jejak untuk`eventName = InvokeWebFunctionEndpoint`. Verifikasi fungsi ARN, waktu acara, ID permintaan, informasi pemanggil yang tersedia untuk jalur otentikasi, dan bidang kesalahan bersyarat apa pun. Jika pemilih berfungsi, kueri mengembalikan `InvokeWebFunctionEndpoint` peristiwa untuk permintaan pengujian Anda, mirip dengan contoh dithe section called “Contoh: pemanggilan yang berhasil (pemanggil yang tidak diautentikasi)”.

Sebagai gantinya, untuk merekam peristiwa data di penyimpanan data peristiwa CloudTrail Lake, tambahkan pemilih bidang yang sama ke penyimpanan data peristiwa. Untuk informasi selengkapnya, silakan lihat Panduan Pengguna AWS CloudTrail .

Bidang dalam suatu InvokeWebFunctionEndpoint acara

Tabel berikut menjelaskan bidang dalam suatu `InvokeWebFunctionEndpoint` peristiwa.

Bidang	Catatan
`eventVersion`	Versi format peristiwa.
`eventSource`	Selalu`lambda.amazonaws.com` .
`eventName`	Selalu`InvokeWebFunctionEndpoint` .
`eventCategory`	Selalu`Data`.
`eventType`	Jenis acara, misalnya`AwsApiCall` .
`readOnly`	Selalu`false`, karena pemanggilan menjalankan kode fungsi Anda.
`managementEvent`	Selalu`false`.
`eventTime`	Tanggal dan waktu, di UTC, Lambda merekam acara tersebut. Untuk pemanggilan yang berhasil, Lambda merekam peristiwa saat fungsi Anda memulai responsnya, sehingga `eventTime` bisa lebih lambat dari saat permintaan tiba.
`sourceIPAddress`	Alamat IP di mana permintaan itu dibuat.

Bidang	Catatan
`userAgent`	Agen yang digunakan penelepon untuk membuat permintaan. Hadir saat penelepon mengirim `User-Agent` header.
`requestParameters`	Berisi`endpointArn`, ARN dari titik akhir yang dipanggil. Muatan permintaan tidak dicatat.
`responseElements`	Selalu`null`.
`resources`	Berisi satu entri dengan `accountId` `type`,, dan ARN fungsi.
`additionalEventData`	Berisi`functionRevisionArn`, ARN dari revisi yang melayani permintaan. Bidang ini hadir di setiap acara, termasuk penolakan.
`tlsDetails`	Versi TLS, cipher suite, dan header host permintaan.
`userIdentity`	Identitas penelepon. Untuk detailnya, lihat daftar berikut.
`sharedEventID`	Hadir saat CloudTrail memberikan acara yang sama ke lebih dari satu Akun AWS.
`errorCode`, `errorMessage`	Hadir hanya pada peristiwa kesalahan.
`requestID`, `eventID`, `recipientAccountId`, `awsRegion`	CloudTrail Bidang standar yang mengidentifikasi permintaan, acara, penerima Akun AWS, dan Wilayah AWS.

Peristiwa Web Function menggunakan bentuk ARN berikut:

- Fungsi (`resources[].ARN`) — `arn:aws:lambda:us-east-1:123456789012:web-function/my-fn`
- Titik akhir (`requestParameters.endpointArn`) — `arn:aws:lambda:us-east-1:123456789012:web-function/my-fn/endpoint/my-endpoint`
- Revisi (`additionalEventData.functionRevisionArn`) — `arn:aws:lambda:us-east-1:123456789012:web-function/my-fn/revision/rev-a1b2c`

`userIdentity`Bidang tergantung pada bagaimana pemanggil diautentikasi:

- Pemanggilan yang tidak diautentikasi direkam dengan tipe. `userIdentity` `Unknown`
- IAM-authenticated penelepon dicatat dengan kepala sekolah mereka sendiri. Misalnya, penelepon yang menggunakan kredensyal sementara muncul dengan tipe`AssumedRole`, an`arn`, dan a. `sessionContext`
- `accessKeyId`Bidang tidak dihuni.

Peristiwa kesalahan

CloudTrail juga mencatat upaya pemanggilan yang gagal setelah pemanggil diautentikasi. Tabel berikut menjelaskan peristiwa kesalahan ini.

Skenario	ERRORCODE	errorMessage
Penelepon tidak diotorisasi	`AccessDenied`	Pengguna tidak berwenang untuk melakukan: `lambda:InvokeWebFunctionEndpoint`
Tingkat permintaan, konkurensi, atau batas vCPU tercapai (titik akhir atau akun)	`ThrottlingException`	Tarif terlampaui
Kesalahan layanan	`ServiceException`	Kesalahan Layanan
Kesalahan aplikasi atau permintaan klien tidak valid	Tidak berlaku	Diperlakukan sebagai panggilan API yang berhasil. Doa mencapai dan menjalankan fungsi.

Doa yang berhasil tidak `errorCode` memiliki atau. `errorMessage` Penolakan otorisasi dicatat sebagai. `AccessDenied` CloudTrail tidak mencatat permintaan yang gagal otentikasi, seperti permintaan dengan tanda tangan yang hilang, cacat, tidak valid, atau kedaluwarsa, dan tidak mencatat permintaan yang ditolak Lambda sebelum otentikasi, seperti permintaan ke titik akhir yang 0. `rateLimit` Kegagalan infrastruktur otentikasi internal juga tidak dicatat. Untuk mendeteksi lonjakan permintaan yang ditolak, alarm menyala `HTTPResponseStatusClass4XX` di `AWS/Lambda/Web` namespace.

Contoh: pemanggilan yang berhasil (pemanggil yang tidak diautentikasi)

Contoh berikut menunjukkan pemanggilan yang berhasil oleh pemanggil yang tidak diautentikasi.


```
{
    "Records": [
        {
            "eventVersion": "1.11",
            "userIdentity": {
                "type": "Unknown"
            },
            "eventTime": "2026-09-22T18:41:07Z",
            "eventSource": "lambda.amazonaws.com",
            "eventName": "InvokeWebFunctionEndpoint",
            "awsRegion": "us-east-1",
            "sourceIPAddress": "203.0.113.42",
            "userAgent": "curl/8.4.0",
            "requestParameters": {
                "endpointArn": "arn:aws:lambda:us-east-1:123456789012:web-function/my-
fn/endpoint/my-endpoint"
            },
            "responseElements": null,
            "additionalEventData": {
                "functionRevisionArn": "arn:aws:lambda:us-east-1:123456789012:web-
function/my-fn/revision/rev-a1b2c"
            },
            "requestID": "b1f8a4c2-7e3d-4a19-9f6b-2c5d8e0a1234",
            "eventID": "d4c3b2a1-5f6e-7a8b-9c0d-1e2f3a4b5c6d",
            "readOnly": false,
            "resources": [
                {
                    "accountId": "123456789012",
                    "type": "AWS::Lambda::WebFunction",
                    "ARN": "arn:aws:lambda:us-east-1:123456789012:web-function/my-fn"
                }
            ],
            "eventType": "AwsApiCall",
            "managementEvent": false,
            "recipientAccountId": "123456789012",
            "eventCategory": "Data",
            "tlsDetails": {
                "tlsVersion": "TLSv1.3",
                "cipherSuite": "TLS_AES_128_GCM_SHA256",
```

```
                "clientProvidedHostHeader": "bcdfghjklmnpqrst.b7t.lambda-web.us-
east-1.on.aws"
            }
        }
    ]
}
```

Untuk IAM-authenticated pemanggil, `userIdentity` bidang tersebut malah membawa prinsipal pemanggil, misalnya ketik `AssumedRole` dengan an `arn` dan a`sessionContext`, dan bidang yang tersisa tidak berubah.

Contoh: doa ditolak

Peristiwa kesalahan membawa `errorCode` dan`errorMessage`. Contoh berikut menunjukkan pemanggilan yang ditolak.

```
{
    "Records": [
        {
            "eventVersion": "1.11",
            "userIdentity": {
                "type": "AssumedRole",
                "principalId": "ARO01EXAMPLEID:session-name",
                "arn": "arn:aws:sts::123456789012:assumed-role/app-invoke-role/session-
name",
                "accountId": "123456789012",
                "sessionContext": {
                    "sessionIssuer": {
                        "type": "Role",
                        "principalId": "ARO01EXAMPLEID",
                        "arn": "arn:aws:iam::123456789012:role/app-invoke-role",
                        "accountId": "123456789012",
                        "userName": "app-invoke-role"
                    },
                    "attributes": {
                        "creationDate": "2026-09-22T18:39:55Z",
                        "mfaAuthenticated": "false"
                    }
                }
            },
            "eventTime": "2026-09-22T18:42:13Z",
            "eventSource": "lambda.amazonaws.com",
            "eventName": "InvokeWebFunctionEndpoint",
```

```
            "awsRegion": "us-east-1",
            "sourceIPAddress": "203.0.113.42",
            "userAgent": "aws-sdk-python/1.34.0",
            "errorCode": "AccessDenied",
            "errorMessage": "User is not authorized to perform:
 lambda:InvokeWebFunctionEndpoint",
            "requestParameters": {
                "endpointArn": "arn:aws:lambda:us-east-1:123456789012:web-function/my-
fn/endpoint/my-endpoint"
            },
            "responseElements": null,
            "additionalEventData": {
                "functionRevisionArn": "arn:aws:lambda:us-east-1:123456789012:web-
function/my-fn/revision/rev-a1b2c"
            },
            "requestID": "c2e9b5d3-8f4e-4b2a-a07c-3d6e9f1b2345",
            "eventID": "e5d4c3b2-6a7f-8b9c-0d1e-2f3a4b5c6d7e",
            "readOnly": false,
            "resources": [
                {
                    "accountId": "123456789012",
                    "type": "AWS::Lambda::WebFunction",
                    "ARN": "arn:aws:lambda:us-east-1:123456789012:web-function/my-fn"
                }
            ],
            "eventType": "AwsApiCall",
            "managementEvent": false,
            "recipientAccountId": "123456789012",
            "eventCategory": "Data",
            "tlsDetails": {
                "tlsVersion": "TLSv1.3",
                "cipherSuite": "TLS_AES_128_GCM_SHA256",
                "clientProvidedHostHeader": "bcdfghjklmnpqrst.b7t.lambda-web.us-
east-1.on.aws"
            }
        }
    ]
}
```

Acara manajemen di CloudTrail

CloudTrail merekam peristiwa manajemen Fungsi Web secara default. Anda dapat melihat 90 hari terakhir acara manajemen dalam riwayat CloudTrail Acara tanpa membuat jejak. Untuk membuatnya

lebih lama, buat penyimpanan data acara trail atau CloudTrail Lake. Untuk informasi selengkapnya, lihat Pencatatan peristiwa manajemen di Panduan AWS CloudTrail Pengguna.

CloudTrail mencatat operasi Fungsi Web berikut sebagai peristiwa manajemen. `eventName`Bidang setiap acara sama dengan nama operasi.

Sumber daya atau tugas	Operasi
Fungsi Web	`CreateWebFunction` , `GetWebFunction` , `ListWebFunctions` , `DeleteWebFunction`
Revisi	`CreateWebFunctionRevision` , `GetWebFunctionRevision` , `ListWebFunctionRevisions` , `DeleteWebFunctionRevision`
Titik akhir	`CreateWebFunctionEndpoint` , `GetWebFunctionEndpoint` , `ListWebFunctionEndpoints` , `UpdateWebFunctionEndpoint` , `DeleteWebFunctionEndpoint`
Pengaturan akun	`GetWebAccountSettings`
Kebijakan sumber daya	`PutResourcePolicy` , `GetResourcePolicy` , `DeleteResourcePolicy`
Tanda	`TagResource` , `UntagResource` , `ListTags`

Pemanggilan Fungsi Web adalah operasi data-plane. CloudTrail mencatatnya hanya sebagai peristiwa data, seperti yang dijelaskan dalamthe section called “Peristiwa data Fungsi Web di CloudTrail”.

Sumber daya terkait

Untuk informasi selengkapnya, lihat sumber daya berikut:

- the section called “CloudTrail log”
- the section called “Izin”
- Membuat jejak untuk Anda Akun AWS

Aplikasi contoh Fungsi Web

Setiap sampel digunakan `index.js` sebagai file entri dan mendengarkan pada host dan port dari`AWS_LAMBDA_HTTP_ENDPOINT`. Package aplikasi dengan dependensi produksinya dan terapkan dari objek Amazon S3 berversi.

Tabel berikut mencantumkan setiap sampel dan apa yang dapat Anda pelajari darinya.

Sampel	Gunakan untuk belajar
the section called “Minimal Ekspres application”	Mulai server HTTP denganExpress.
the section called “Hono API dan file statis”	Sajikan rute API dan file statis denganHono.
the section called “Server-sent acara”	Streaming peristiwa yang dikirim server dan berhenti bekerja saat klien terputus.
the section called “Multi-Region rute verifikasi”	Kembalikan Wilayah penayangan saat memvalidasi perutean Multi-wil ayah.

Minimal Ekspres application

Sampel berbasis kerangka kerja ini merupakan alternatif dari starter ketergantungan nol yang dibuat oleh. `--hello-world` Untuk penyebaran starter, lihatthe section called “Menyebarkan dan menguji Fungsi Web Anda”.

Instal Express:

```
npm init -y
npm install express
```

Buat`index.js`:

```
const express = require('express');
const app = express();
const [host, port] = (process.env.AWS_LAMBDA_HTTP_ENDPOINT ||
 'localhost:3000').split(':');

app.get('/', (request, response) => {
  response.send('Hello from Lambda Web Functions!');
});

app.listen(port, host);
```

Untuk panduan pengemasan, lihatthe section called “Package dan menyebarkan sampel”.

Hono API dan file statis

Instal dependensi:

```
npm init -y
npm pkg set type=module
npm install hono @hono/node-server
```

Buat`index.js`:

```
import { serve } from '@hono/node-server';
import { serveStatic } from '@hono/node-server/serve-static';
import { Hono } from 'hono';
import { secureHeaders } from 'hono/secure-headers';

const app = new Hono();
app.use(secureHeaders());
app.use('/static/*', serveStatic({ root: './' }));
app.get('/api/health', (context) => context.json({ status: 'ok' }));
app.notFound((context) => context.json({ error: 'not found' }, 404));

const [host, port] = (process.env.AWS_LAMBDA_HTTP_ENDPOINT ||
 'localhost:3000').split(':');

serve({ fetch: app.fetch, port: Number(port), hostname: host });
```

Letakkan file statis di bawah `static` direktori dan sertakan direktori dalam paket penyebaran.

Server-sent acara

Tambahkan rute berikut ke Express aplikasi:

```
app.get('/events', (request, response) => {
  response.setHeader('Content-Type', 'text/event-stream');
  response.setHeader('Cache-Control', 'no-cache');
  response.flushHeaders();

  const interval = setInterval(() => {
    response.write(`data: ${JSON.stringify({ time: Date.now() })}\n\n`);
  }, 1000);

  response.on('close', () => {
    clearInterval(interval);
  });
});
```

Permintaan tetap aktif sampai aplikasi mengakhiri respons atau klien terputus. Tetapkan batas waktu revisi untuk durasi streaming maksimum. WebSocketupgrade tidak didukung.

Multi-Region rute verifikasi

Tambahkan rute yang mengembalikan Wilayah tempat permintaan dieksekusi:

```
app.get('/region', (request, response) => {
  response.json({
    region: process.env.AWS_REGION,
    requestId: request.get('lambda-web-request-id')
  });
});
```

Panggil rute dari lokasi klien yang penting bagi aplikasi Anda. Untuk `MultiRegion` titik akhir, Wilayah yang dipilih berasal dari perutean berbasis latensi dan dapat berubah. Jangan gunakan lingkungan eksekusi sebagai penyimpanan sesi.

Package dan menyebarkan sampel

```
npm ci --omit=dev
zip -r function.zip index.js package.json package-lock.json node_modules
```

```
aws s3 cp function.zip s3://amzn-s3-demo-bucket/samples/function.zip
```

Sertakan file lain yang digunakan sampel yang dipilih, seperti `static` direktori dalam Hono sampel. Untuk membuat revisi yang mereferensikan objek yang diunggah, lihat the section called “Kode di Amazon S3” dan. the section called “Revisi” Untuk `deploy` alur kerja tingkat yang lebih tinggi, lihat. the section called “Membuat Fungsi Web (AWS CLI)” Setelah revisi `Active` dan titik akhir menyajikannya, kirim permintaan ke domain titik akhir untuk mengonfirmasi respons sampel. Untuk cakupan tes di luar tes asap ini, lihatthe section called “Pengujian”.

Praktik terbaik untuk Fungsi Web Lambda

Amankan setiap titik akhir

Setiap titik akhir dapat dijangkau dari internet, jadi bagaimana Anda mengautentikasi penelepon, memasang pintu depan, dan izin ruang lingkup menentukan siapa yang dapat menjangkau aplikasi Anda dan apa yang dapat diakses. Gunakan praktik berikut:

- Setiap titik akhir dapat dijangkau dari internet, apa pun jenis otentikasinya. Dengan`ApplicationManaged`, Lambda tidak mengautentikasi permintaan, sehingga setiap klien yang mengetahui domain dapat memanggil aplikasi Anda. Menerapkan otentikasi, otorisasi, validasi input, pemeriksaan ukuran permintaan, dan perlindungan penyalahgunaan dalam aplikasi atau di pintu depan.
- Gunakan `IamAuth` saat penelepon dapat menandatangani permintaan dan kebijakan IAM menyediakan model otorisasi yang diperlukan. Kebijakan sumber daya lingkup ke ARN dan prinsipal titik akhir tertentu.
- Letakkan Amazon CloudFront atau Amazon API Gateway di depan titik akhir saat Anda memerlukan domain khusus, AWS WAF inspeksi, caching, atau kontrol tepi terpusat. Batasi asal sehingga klien tidak dapat melewati pintu depan. Dengan CloudFront, gunakan `IamAuth` endpoint dan origin access control (OAC), seperti yang dijelaskan dalam. the section called “CloudFront integrasi” Dengan pintu depan lainnya, titik akhir tetap dapat dijangkau secara langsung, jadi verifikasi rahasia atau token bersama dalam aplikasi.
- Gunakan peran eksekusi khusus dengan izin hak istimewa paling sedikit. Pisahkan identitas penerapan, pengunggah kode, prinsip layanan Lambda, peran eksekusi, dan izin pemanggil titik akhir.
- Jangan menaruh rahasia dalam paket penyebaran atau log. Muat rahasia saat runtime dari toko rahasia yang sesuai, cache hanya selama diperlukan, dan edit bidang permintaan sensitif.

Pertahankan artefak penyebaran yang tidak dapat diubah

- Aktifkan pembuatan versi pada bucket kode dan rekam ID versi objek untuk setiap penerapan.
- Jangan menimpa atau menghapus versi objek yang digunakan oleh revisi aktif atau rollback. Simpan akses utama layanan Lambda selama revisi mereferensikan objek.
- Bangun paket penerapan dalam pipeline yang dapat diulang. Instal dependensi produksi dari file kunci, kecualikan file pengembangan, dan uji arsip yang tepat sebelum mengunggah.
- Bangun dependensi asli untuk Arm64. Modul asli yang dibuat untuk x86_64 tidak berjalan di lingkungan eksekusi Fungsi Web.
- Gunakan bucket, awalan, peran, atau akun terpisah saat lingkungan memiliki persyaratan kepercayaan atau retensi yang berbeda.

Desain untuk permintaan bersamaan

- Simpan data khusus permintaan dalam lingkup permintaan. Jangan menyimpan kredensi, data pelanggan, objek respons, atau status per permintaan yang dapat berubah dalam variabel global.
- Gunakan asinkron I/O. CPU sinkron atau kerja sistem file memblokir loop Node.js peristiwa dan menunda setiap permintaan dalam penerbangan di lingkungan.
- Gunakan `worker_threads` untuk CPU-intensive pekerjaan yang harus menggunakan beberapa vCPU, atau `maxConcurrencyPerEnvironment` lebih rendah untuk mengurangi pertengkaran.
- Akun untuk Node.js heap secara terpisah dari memori lingkungan. Lingkungan memiliki 2 GB, sedangkan tumpukan V8 default adalah 1.024 MB. Jika Anda meningkatkan heap dengan `--max-old-space-size` in`NODE_OPTIONS`, tinggalkan memori untuk buffer, modul asli, runtime, dan permintaan bersamaan.
- Gunakan kembali klien yang diinisialisasi dan kumpulan koneksi di seluruh permintaan, tetapi validasi koneksi basi dan pertumbuhan cache terikat. Pertahankan status yang diperlukan di luar lingkungan eksekusi.
- Jaga agar koneksi keluar tetap hidup sehingga permintaan ke database dan dependensi lain menggunakannya kembali alih-alih membuka koneksi baru setiap kali. Node.jsmemungkinkan keep-alive secara default di agen HTTP globalnya. Jika Anda membuat agen sendiri atau mengonfigurasi klien SDK, tetap aktifkan keep-alive. Juga menangani koneksi yang ditutup oleh sisi jarak jauh saat mereka menganggur.

- Gunakan nama unik untuk file sementara dan hapus ketika mereka tidak lagi diperlukan. Beberapa permintaan berbagi sistem `/tmp` file 512 MB lingkungan, dan file dapat bertahan di seluruh permintaan hingga lingkungan dihapus.

Jaga agar startup dapat diprediksi

- Mulai server HTTP segera dan dengarkan nilai yang tepat dari. `AWS_LAMBDA_HTTP_ENDPOINT`
- Jaga agar paket tetap kecil dan hapus dependensi produksi yang tidak digunakan.
- Inisialisasi hanya sumber daya yang dibutuhkan sebelum server dapat menerima lalu lintas. Lakukan pekerjaan opsional dengan malas, tetapi mencegah permintaan bersamaan menduplikasi inisialisasi yang mahal.
- Tangani `SIGTERM` pembersihan yang selesai dalam 1 detik, tetapi jangan mengandalkan panggilan balik shutdown untuk mempertahankan data yang diperlukan.

Tetapkan batas dari perilaku terukur

- Uji beban dengan dependensi aplikasi yang sama dan layanan hilir yang digunakan dalam produksi. Tingkatkan konkurensi hingga latensi, konsumsi CPU, memori, atau kapasitas hilir menjadi kendala pengikatan.
- Gunakan `maxConcurrencyPerEnvironment` untuk mengikat pertikaian per proses, `maxEnvironments` ke skala horizontal terikat, dan `rateLimit` untuk melindungi dependensi hilir dari semburan permintaan.
- Periksa akun yang diterapkan dan batas Regional sebelum pengujian kinerja dan peluncuran. Konfigurasi titik akhir tidak dapat membuat kapasitas di luar batas akun.
- Coba lagi respons HTTP 429 dan respons HTTP 503 yang tidak memiliki header. `lambda-web-error-type` Gunakan backoff dan jitter, dan coba lagi hanya ketika operasi idempoten. Gunakan kunci idempotensi atau deduplikasi tingkat aplikasi untuk penulisan.

Terapkan dengan verifikasi eksplisit dan rollback

- Gunakan bobot revisi manual untuk titik akhir produksi. Simpan penerapan revisi terbaru otomatis untuk lingkungan pengembangan di mana peluncuran langsung dapat diterima.

- Tunggu hingga revisi baru selesai `Active` dan pembaruan titik akhir selesai sebelum Anda mengevaluasinya. Untuk `MultiRegion` dan `PerRegion` titik akhir, verifikasi status pembaruan di setiap Wilayah.
- Gunakan log aplikasi atau metrik khusus yang menyertakan pengenal revisi selama penerapan canary. Service-vended metrik tidak termasuk dimensi revisi.
- Sebelum Anda mengalihkan lalu lintas, tentukan metrik yang harus dipenuhi oleh revisi baru dan berapa lama Anda mengamatinya (kriteria panggang), dan ambang batas yang memicu kemunduran. Pertahankan versi objek kode revisi sebelumnya, akses kunci, dan konfigurasi hingga jendela rollback berakhir. Lambda secara otomatis menghapus revisi tertua yang tidak dirutekan titik akhir ketika Fungsi Web melebihi batas revisinya, jadi pertahankan revisi rollback pada titik akhir (misalnya, titik akhir pengujian) atau siap untuk membuatnya kembali dari versi objek yang sama. Lihat perinciannya di the section called “Batas revisi dan pembersihan otomatis”.

Membangun baseline operasional

- Memancarkan log aplikasi terstruktur dengan ID permintaan, ID revisi, rute, kode status, durasi, dan pengidentifikasi yang diperlukan untuk mendiagnosis kegagalan hilir. Jangan mencatat kredensi, token, atau badan permintaan sensitif.
- Alarm pada respons 5xx dan throttle, dan pantau permintaan yang dieksekusi, lingkungan aktif, dan kapasitas akun. Konfigurasikan metrik kesalahan jarang untuk menangani data yang hilang dengan tepat.
- Gunakan ID permintaan respons dan header log-stream untuk menghubungkan kegagalan klien dengan log. Aktifkan log sistem DEBUG hanya saat Anda membutuhkan acara tambahan.
- Rekam operasi manajemen dan peristiwa data pemanggilan titik akhir AWS CloudTrail saat persyaratan audit Anda membutuhkannya.

Beroperasi MultiRegion dan PerRegion titik akhir sebagai sistem terdistribusi

- Buat data aplikasi, rahasia, kunci, dan dependensi hilir tersedia di setiap Wilayah yang dapat menerima lalu lintas.
- Uji kegagalan penerapan sebagian dan kegagalan ketergantungan Regional. Jangan berasumsi bahwa semua Wilayah menjalankan revisi yang sama sampai setiap pembaruan Regional melaporkan keberhasilan.

- Setelah setiap pembaruan titik akhir, periksa `updateStatus` setiap entri di`regionalEndpoints`. Replikasi dapat gagal di beberapa Wilayah sementara berhasil di daerah lain. Akibatnya, pembaruan dapat diterapkan sebagian. Coba lagi atau putar kembali sebelum Anda memperlakukan penerapan sebagai selesai.
- Untuk `PerRegion` titik akhir, uji DNS eksternal atau failover pintu depan Anda secara independen dari penerapan Fungsi Web.
- Memusatkan pandangan operasional di Wilayah asal, tetapi pertahankan jalur pemulihan untuk penundaan atau gangguan telemetri.

Menguji Fungsi Web Lambda

Fungsi Web menjalankan server HTTP standar, sehingga Anda dapat menguji sebagian besar perilaku aplikasi secara lokal dengan alat normal kerangka kerja Anda. Pengujian cloud masih diperlukan untuk memvalidasi otorisasi IAM, akses Amazon S3 dan AWS Key Management Service (AWS KMS), status siklus hidup Fungsi Web, penskalaan, telemetri, dan perilaku Multi-wilayah.

Lapisan uji yang direkomendasikan

Tabel berikut menjelaskan setiap lapisan pengujian, apa yang divalidasi, dan di mana ia berjalan.

Lapisan uji	Apa yang divalidasi	Dimana ia berjalan
Unit	Logika bisnis, validasi, keputusan otorisasi, serialisasi, dan penanganan kesalahan.	Pembangunan lokal dan membangun lingkungan.
Integrasi HTTP	Rute, middleware, header, streaming, batas waktu, pemutusan yang anggun, dan perilaku kerangka kerja.	Terhadap server produksi yang berjalan secara lokal.
Paket	File entri, dependensi produksi, tata letak arsip, startup, dan penggunaan server. `AWS_LAMBDA_HTTP_ENDPOINT`	Dari arsip persis yang diunggah oleh pipeline penerapan.
Integrasi cloud	IAM, kebijakan bucket, kunci, status siklus hidup, perilaku titik akhir, telemetri, dan panggilan layanan hilir. AWS	Dalam lingkungan yang terisolasi Akun AWS atau pengujian.

Lapisan uji	Apa yang divalidasi	Dimana ia berjalan
Beban dan ketahanan	Keamanan konkurensi, latensi, pelambatan, batas hilir, rollback, dan perilaku kegagalan Regional.	Dalam lingkungan cloud seperti produksi dengan biaya eksplisit dan kontrol lalu lintas.

Jalankan server HTTP secara lokal

Gunakan fallback lokal untuk`AWS_LAMBDA_HTTP_ENDPOINT`, seperti yang ditunjukkan dalam file. the section called “Minimal Ekspres application” Anda juga dapat mengatur variabel secara eksplisit untuk menguji jalur startup yang sama yang digunakan oleh Lambda:

```
AWS_LAMBDA_HTTP_ENDPOINT=127.0.0.1:3000 node index.js
```

Kirim permintaan dengan klien HTTP normal Anda dan verifikasi kode status, badan respons, header, validasi permintaan, middleware otentikasi, batas waktu, streaming, dan penanganan pemutusan klien.

Jalankan permintaan bersamaan selama pengujian integrasi lokal. Periksa status permintaan global, cache yang dapat diubah bersama, inisialisasi duplikat, tabrakan nama file, dan pemblokiran loop peristiwa.

Uji paket penerapan

1. Instal dependensi dari file kunci yang dikomit dan buat arsip menggunakan perintah yang sama dengan pipeline penerapan.
2. Ekstrak arsip ke direktori bersih. Jalankan aplikasi dari direktori itu tanpa akses ke dependensi pengembangan atau file sumber yang tidak dilacak.
3. Verifikasi bahwa file entri yang dikonfigurasi ada di jalur yang diharapkan dan mulai server HTTP.
4. Rekam checksum arsip dan ID versi objek Amazon S3 yang digunakan oleh revisi. Gunakan nilai-nilai tersebut untuk mengidentifikasi artefak yang tepat yang sedang diuji.

Fungsi Web berjalan di Arm64. Ketika sebuah paket berisi dependensi asli, buat dan uji dependensi tersebut untuk Linux Arm64 daripada menggunakan kembali artefak dari komputer pengembangan x86-64. Misalnya, Anda dapat menginstal dependensi produksi dalam wadah Arm64:

```
docker run --rm --platform linux/arm64 \
    --volume "$PWD":/app \
    --workdir /app \
    node:24 \
    npm ci --omit=dev
```

Uji di lingkungan cloud yang terisolasi

1. Unggah arsip yang diuji ke bucket Amazon S3 berversi dan buat revisi yang mereferensikan versi objek yang dihasilkan.
2. Polling revisi sampai menjadi `Active` atau`Failed`. Gagal tes dengan yang dilaporkan `stateReason` alih-alih menunggu tanpa batas waktu.
3. Buat atau perbarui titik akhir pengujian. Polling keduanya `state` dan`updateStatus`. Untuk `MultiRegion` dan `PerRegion` titik akhir, verifikasi setiap status Regional.
4. Kirim permintaan ke titik akhir dan verifikasi respons aplikasi, log`lambda-web-request-id`, metrik, dan efek samping hilir yang diharapkan.
5. Hapus titik akhir, fungsi, objek uji, grup log, dan sumber daya pengujian lainnya saat pengujian selesai.

Uji otorisasi dan eksposur

- Untuk`IamAuth`, verifikasi bahwa permintaan yang tidak ditandatangani gagal, prinsipal yang diizinkan berhasil, prinsip yang ditolak gagal, dan akses lintas akun memerlukan kebijakan identitas dan sumber daya.
- Untuk`ApplicationManaged`, uji kredenal aplikasi yang hilang, kedaluwarsa, cacat, dan tidak mencukupi. Uji otorisasi secara terpisah dari otentikasi.
- Uji permintaan preflight browser, asal-usul yang diizinkan dan ditolak, permintaan yang dikreditkan, dan header keamanan saat browser memanggil titik akhir.
- Ketika pintu depan melindungi titik akhir, verifikasi bahwa permintaan asal langsung ditolak dan bahwa hanya identitas pintu depan atau jalur jaringan yang diharapkan yang dapat memanggilnya.

Uji konkurensi, penskalaan, dan pelambatan

- Tingkatkan beban secara bertahap saat merekam latensi, respons 4xx dan 5xx, throttle, lingkungan aktif, dan pemanfaatan hilir.

- Uji yang dikonfigurasi`rateLimit`,`maxEnvironments`, dan `maxConcurrencyPerEnvironment` secara independen. Gunakan header `lambda-web-error-type` respons (`Platform.EndpointRequestThrottle`,,, `Platform.AccountRequestThrottle` `Platform.EndpointMaxEnvThrottlePlatform.AccountActiveVCPUArm64Throttle`, atau`Platform.EndpointConcurrentInitThrottle`) untuk mengonfirmasi batas mana yang menghasilkan setiap respons HTTP 429. Rekam respons HTTP 503 yang tidak memiliki `lambda-web-error-type` header secara terpisah. Tanggapan ini dapat menunjukkan pelepasan beban sementara atau kesalahan internal. Lihat perinciannya di the section called “Jenis kesalahan Fungsi Web”.
- Uji semburan lalu lintas dan start dingin, tidak hanya throughput kondisi mapan.
- Gunakan data uji unik dan kunci idempotensi. Jangan mencoba lagi permintaan non-idempoten secara otomatis selama uji beban.
- Tetapkan lalu lintas eksplisit, durasi, dan batas biaya sebelum memulai uji beban. Tes koordinat yang dapat memengaruhi kuota akun bersama atau sistem hilir.

Uji penerapan dan rollback

Dalam lingkungan pengujian, gunakan dua revisi dan lakukan prosedur perutean tertimbang yang sama yang digunakan dalam produksi. Verifikasi bahwa log atau metrik kustom Anda dapat mengatribusikan permintaan ke setiap revisi. Picu kondisi rollback dengan sengaja, kembalikan semua lalu lintas ke revisi sebelumnya, dan konfirmasikan bahwa objek kode dan dependensinya tetap tersedia.

Untuk `PerRegion` titik akhir `MultiRegion` atau titik akhir, uji kegagalan pembaruan sebagian dan konfirmasikan bahwa otomatisasi penerapan mendeteksi status Regional yang gagal. Uji tindakan pemulihan dan ulangi pembaruan titik akhir setelah memperbaiki penyebabnya.

Memecahkan Masalah Fungsi Web Lambda

Mulailah dengan status sumber daya, respons HTTP, dan log untuk permintaan. Fungsi Web mengekspos status terpisah untuk fungsi, revisi, titik akhir, dan setiap Wilayah atau titik akhir. `MultiRegion` `PerRegion` Kegagalan dalam satu sumber daya tidak selalu mengubah keadaan yang lain.

> **Note**
>
> Sebelum Anda menjalankan perintah diagnostik dalam topik ini, pastikan Anda memiliki AWS CLI akses dan izin untuk memanggil operasi Fungsi `get-*` Web Lambda.

Kumpulkan informasi diagnostik

1. Dapatkan Fungsi Web, revisi, dan titik akhir. Catat masing-masing `statestateReason`,,`updateStatus`, dan `updateStatusReason` nilai.

   ```
   aws lambda-web get-web-function \
       --function-name my-web-app

   aws lambda-web get-web-function-revision \
       --function-name my-web-app \
       --revision-id revision-id

   aws lambda-web get-web-function-endpoint \
       --function-name my-web-app \
       --endpoint-name prod
   ```

   Ganti *my-web-app* dengan nama fungsi Anda, *revision-id* dengan revisi untuk memeriksa, dan *prod* dengan nama endpoint Anda.

2. Kirim permintaan dan catat kode status dan header respons.

   ```
   curl --include https://endpoint-domain/path
   ```

   `lambda-web-request-id`Header mengidentifikasi permintaan. Saat ada, `lambda-web-logstream` header mengidentifikasi aliran CloudWatch log Amazon Logs yang menangani permintaan, dan `lambda-web-error-type` header mengidentifikasi kesalahan platform.

3. Tinjau log aplikasi dan sistem di grup log terpusat di Wilayah rumah Fungsi Web. Untuk nama grup log dan panduan korelasi, lihatthe section called “Beberapa catatan”.

4. Untuk `MultiRegion` dan `PerRegion` titik akhir, periksa setiap entri di. `regionalEndpoints` Status tingkat titik akhir yang sukses tidak berarti bahwa pembaruan terbaru berhasil di setiap Wilayah.

Masalah penyebaran dan startup

Gejala	Apa yang harus diperiksa	Resolusi
Sebuah revisi adalah `Failed`	Revisi`stateReason` , versi objek Amazon S3 yang direferensikan, kebijakan bucket, peran eksekusi, runtime, dan kunci. AWS KMS	Perbaiki konfigurasi yang mendasarinya dan buat revisi baru. Revisi tidak dapat diubah, dan revisi yang gagal tidak kembali ke. `Active`
Titik akhirnya adalah`Active`, tetapi permintaan gagal selama startup aplikasi	Log sistem dan aplikasi untuk kesalahan entry-file, kesalahan ketergantungan, atau server yang tidak pernah mendengarkan. `AWS_LAMBDA_HTTP_ENDPOINT`	Jalankan aplikasi yang dikemas secara lokal, verifikasi file entri dan dependensi produksi, dan dengarkan host dan port yang tepat yang disediakan Lambda. Terapkan koreksi sebagai revisi baru.
Pembaruan titik akhir gagal	`updateStatusReason` pada titik akhir dan, untuk `MultiRegion` dan `PerRegion` titik akhir, pada setiap entri Regional.	Perbaiki ketergantungan yang dilaporkan atau kegagalan Regional, lalu panggil `UpdateWebFunctionEndpoint` lagi untuk mencoba lagi pembaruan.
`does not have s3:GetObject` pada `create-web-function-revision`	Kode dibaca dari Amazon S3, dan kebijakan bucket atau kebijakan identitas pemanggil tidak memiliki akses baca.	Hibah `s3:GetObject` dan `s3:GetObjectVersion` dalam kebijakan bucket (untuk layanan Lambda) dan kebijakan identitas penelepon.
`AccessDenied` pada `get-web-function-endpoint`	Kebijakan IAM `Resource` tidak termasuk titik akhir.	Termasuk `web-function/` *`name`*`/*` dalam kebijakan`Resource`.

Untuk menghindari kegagalan penerapan berulang, validasi konfigurasi dan startup dalam revisi pengujian sebelum Anda membuat revisi produksi.

Kesalahan CLI dan paket

Kesalahan atau gejala	Penyebab	Resolusi
`invalid choice: lambda-web`	Yang diinstal AWS CLI mendahului perintah Fungsi Web.	Perbarui AWS CLI dan verifikasi yang aws lambda-web help mencantumkan operasi Fungsi Web.
`Unknown parameter: BuildConfig`	Permintaan menggunakan nama PascalCase anggota.	Gunakan anggota permintaan CamelCase yang ditunjukkan dalam panduan ini, `buildConfig` seperti`codeConfig` ,, dan. `serviceConfig`
`Cannot find module '/var/task/index.js'`	File entri yang diperlukan hilang dari root arsip.	Letakkan `index.js` di root paket penyebaran. Sertakan setiap file dan dependensi produksi yang diimpornya.
`ERR_MODULE_NOT_FOUND` untuk file aplikasi	Arsip menghilangkan sumber atau file bawaan yang diimpor oleh`index.js`.	Periksa arsip dengan`unzip -l`, tambahkan file yang hilang, dan buat revisi baru.
`Application.TimeoutListenError` , atau HTTP 502 pada permintaan pertama	Aplikasi mendengarkan pada alamat loopback-only, seperti `127.0.0.1` atau`localhost` , atau pada port selain yang masuk. `AWS_LAMBDA_HTTP_ENDPOINT`	Parse `AWS_LAMBDA_HTTP_ENDPOINT` dan dengarkan host dan port yang tepat itu.
`EMFILE` atau `too many open files`	Aplikasi membocorkan soket, file, atau koneksi database di seluruh permintaan bersamaan.	Tutup pegangan per permintaan, kumpulan koneksi terikat, dan pantau pegangan terbuka di bawah konkurensi berkelanjutan.

Kesalahan atau gejala	Penyebab	Resolusi
`JavaScript heap out of memory`	Aplikasi mencapai batas heap V8 default, yang lebih rendah dari batas memori 2 GB lingkungan eksekusi, atau permintaan bersamaan menghabiskan memori lingkungan yang tersisa.	Periksa objek yang dipertahankan dan cache tak terbatas, dan kurangi konkurensi. Jika aplikasi membutuhkan heap yang lebih besar, tetapkan batas tumpukan V8 yang lebih besar dengan in. `--max-old-space-size` `NODE_OPTIONS` Tinggalkan memori untuk buffer, alokasi asli, dan runtime.
`Cannot find moduleERR_MODULE_NOT_FOUND` Ketergantungan/saat runtime	`node_modules` tidak termasuk dalam paket penerapan.	Instal dependensi produksi dan sertakan dalam paket: jalankan`npm ci --omit=dev` , lalu konfirmasikan dengan. `unzip -l function.zip`
File statis mengembalikan HTTP 404	Aplikasi menyelesaikan jalur file relatif terhadap direktori kerja, atau file tidak dikemas.	Selesaikan jalur dari direktori modul (`__dirname` di CommonJS, atau `import.meta.dirname` dalam modul ES), dan konfirmasikan file ada dalam paket dengan`unzip -l`.
Kesalahan impor ESM, seperti “Tidak dapat menggunakan pernyataan impor di luar modul”	`package.json` tidak mendeklarasikan jenis modul.	Tambahkan `"type": "module"` ke`package.json` , atau gunakan CommonJS secara konsisten.

Kegagalan permintaan HTTP

Respons	Penyebab umum	Resolusi
HTTP 400	Klien menggunakan metode HTTP yang tidak didukung atau mengirim permintaan	Gunakan metode yang didukung dan tinjau header dan batasan permintaan. the section called “Perilaku HTTP”

Respons	Penyebab umum	Resolusi
	bahwa titik akhir tidak dapat memproses.	
HTTP 403	`IamAuth`Permintaan gagal otorisasi karena salah satu alasan berikut: • Permintaan tidak ditandatangani. • Permintaan menghilangkan header hash konten yang diperlukan. • Permintaan memiliki tanda tangan yang tidak valid. • Penelepon tidak memiliki izin. Untuk panggilan lintas akun, kebijakan identitas atau kebijakan berbasis sumber daya mungkin hilang.	Konfirmasikan bahwa permintaan menyertakan `X-Amz-Content-SHA256` header yang diperlukan, tinjau tanda tangan, dan verifikasi `lambda:InvokeWebFunctionEndpoint` dalam kebijakan identitas dan sumber daya yang diperlukan. Untuk persyaratan penandatanganan, lihatthe section called “Menandatangani permintaan”. Untuk persyaratan izin, lihatthe section called “Memberikan izin pemanggilan kepada kepala sekolah”.

Respons	Penyebab umum	Resolusi
HTTP 404	Aplikasi tidak menentuka n rute yang diminta, atau klien menggunakan domain endpoint usang. `Host`Header mungkin juga tidak cocok dengan domain endpoint. Jika Anda mengonfigurasi pintu depan seperti Amazon CloudFront untuk meneruska n `Host` header penampil ke asal, Lambda mengembal ikan kesalahan ini dengan `lambda-web-error-t ype` disetel ke. `Platform. InvalidEndpoint`	Verifikasi rute saat ini`domainName` , jalur permintaan, dan aplikasi. Gunakan `lambda-we b-error-type` dan log untuk membedakan respons platform dari respons aplikasi.
HTTP 409	Titik akhir tidak`Active`, atau titik akhir diselesaikan untuk revisi yang tidak. `Active`	Periksa status revisi dan titik akhir. Jika titik akhir adalah`Pending`, tunggu dan coba lagi; jika ya`Failed`, buat titik akhir baru. Untuk revisi, hapus revisi yang gagal atau tidak aktif dari bobot titik akhir, dan buat revisi pengganti bila diperlukan.

Respons	Penyebab umum	Resolusi
HTTP 429	Salah satu dari hal berikut terjadi: • Permintaan melebihi batas tingkat titik akhir. • Titik akhir kehabisan kapasitas lingkungan. • Akun mencapai permintaan Regional atau batas vCPU. • Terlalu banyak lingkungan eksekusi yang diinisialisasi pada saat yang sama untuk titik akhir.	Baca `lambda-web-error-type` header untuk mengidentifikasi batas mana yang diterapkan. Lihat the section called “Jenis kesalahan Fungsi Web”. Tinjau metrik throttle (`EndpointRequestThrottles` ,,, dan`AccountActiveVCPUArm64Throttles`) `AccountRequestThrottles` `EndpointActiveEnvironmentThrottles` , metrik kapasitas aktif, pengaturan titik akhir, dan batas akun yang diterapkan. Pelambatan inisialisasi bersamaan (`Platform.EndpointConcurrentInitThrottle`) tidak menambah metrik throttle. Coba lagi hanya permintaan idempoten , menggunakan backoff dan jitter.
HTTP 502	Aplikasi Anda gagal sebelum mulai melayani permintaan, Lambda tidak dapat menyiapkan lingkungan eksekusi untuk Fungsi Web Anda, atau aplikasi Anda mengembalikan respons yang tidak valid.	Gunakan `lambda-web-error-type` dan log sistem untuk mengidentifikasi penyebabnya. Lihat the section called “Jenis kesalahan Fungsi Web”.
HTTP 503	Aplikasi macet saat menangani permintaan, titik akhir dinonaktifkan, atau Lambda tidak dapat memperoleh kapasitas atau mencapai aplikasi.	Gunakan`lambda-web-error-type` , saat ada, dan log aplikasi dan sistem untuk mengidentifikasi penyebabnya. Lihat the section called “Jenis kesalahan Fungsi Web”. 503 tanpa `lambda-web-error-type` dapat menunjukkan pelepasan beban sementara atau kesalahan internal. Dalam kedua kasus tersebut, coba lagi permintaan idempoten dengan backoff dan jitter.

Respons	Penyebab umum	Resolusi
HTTP 504	Aplikasi tidak merespons dalam batas waktu revisi ()`Runtime.RequestTimeout` .	Periksa latensi revisi `timeoutSeconds` dan hilir.
HTTP 5xx lainnya	Aplikasi mengembalikan status 5xx, atau Lambda mengalami kesalahan internal (`lambda-web-error-type` tidak ada header).	Gunakan ID permintaan, header log-stream, log aplikasi, dan log sistem untuk mengidentifikasi sumber, dan memeriksa kesehatan proses.
Permintaan hang atau waktu habis pada unggahan besar; log menunjukkan 0 byte yang diterima	Pawang mulai menulis tanggapan saat badan permintaan masih tiba. Jalur permintaan Fungsi Web adalah half-duplex: Lambda meneruskan badan permintaan lengkap sebelum memberikan respons aplikasi Anda.	Baca isi permintaan sepenuhnya sebelum Anda menulis tanggapan. Badan di bawah sekitar 1 MB cocok dengan buffer soket dan dapat menutupi ini.

Minta batas header

Dua batasan berlaku untuk header permintaan. Lambda membatasi blok header menjadi 128 KiB dan menolak permintaan yang lebih besar sebelum mencapai fungsi Anda. Server Node.js HTTP dalam fungsi Anda juga membatasi jumlah baris header yang diterimanya. Keduanya mengembalikan HTTP 431 Request Header Fields Too Large, seperti yang ditunjukkan pada tabel berikut. Untuk batas platform, lihatthe section called “Batas header”.

Gejala	Penyebab	Resolusi
HTTP 431 dengan `lambda-web-error-type: Platform.HeaderTooLarge` Tidak ada pemanggil	Kepala permintaan melebihi 131.072 byte (128 KiB), diukur pada kepala permintaan	Kurangi ukuran header permintaan. Batas ini tidak dapat dinaikkan.

Gejala	Penyebab	Resolusi
an dan tidak ada entri log aplikasi.	penuh saat klien mengirimk annya.	
HTTP 431 dengan `lambda-web-request-id` dan `lambda-web-logstream` tetapi tidak`lambda-web-error-type` . Handler tidak berjalan.	Permintaan memiliki lebih dari 1.000 baris header yang diterima oleh Node.js server, termasuk header yang ditambahkan Lambda.	Naikkan `server.maxHeadersCount` kode server Anda, atau kurangi jumlah baris header yang dikirim klien.

Batas ukuran menghitung setiap byte kepala permintaan saat klien mengirimkannya. Ini termasuk baris permintaan, setiap baris header (termasuk `Host` dan header apa pun yang ditambahkan pustaka klien), dan terminator baris. Header yang ditambahkan Lambda tidak dihitung terhadap batas ukuran.

Ketika permintaan melebihi batas ukuran, Lambda menolaknya sebelum mencapai fungsi Anda, sehingga permintaan tidak menghasilkan pemanggilan dan tidak ada entri log aplikasi. Header `lambda-web-error-type` respons mengidentifikasi penyebabnya.

Batas Node.js hitungan menghitung baris header, bukan nama header yang berbeda. Header yang diulang tiga kali dihitung tiga kali. Akibatnya, permintaan yang mengakumulasi `x-forwarded-for` atau melacak header melalui rantai proxy dapat mencapai batas. Tidak ada header tunggal yang perlu terlihat tidak biasa agar hal ini terjadi. Batas diukur pada permintaan yang diterima fungsi Anda, yang mencakup header yang ditambahkan Lambda. Akibatnya, klien yang mengirim sedikit kurang dari 1.000 baris header dapat mencapainya. Ketika itu terjadi, permintaan mencapai lingkungan eksekusi Anda, dan Node.js server menolaknya selama penguraian. Handler Anda tidak berjalan. Respons membawa `lambda-web-request-id` dan `lambda-web-logstream` header, tetapi tidak membawa `lambda-web-error-type` header.

Jika aplikasi Anda harus menerima lebih banyak baris header daripada yang Node.js diizinkan default, tingkatkan Node.js batas dalam kode server Anda. Setel `server.maxHeadersCount` ke jumlah baris header yang akan diterima, atau atur `0` untuk menonaktifkan pemeriksaan Node.js hitungan. Batas ukuran 128 KiB masih berlaku.

```
const server = http.createServer(handler);
server.maxHeadersCount = 2000;
```

```
server.listen(port, host);
```

Anda tidak perlu mengubah batas ukuran Node.js header. Node.jsRuntime terkelola menyetelnya agar sesuai dengan batas platform, sehingga batas platform berlaku untuk ukuran header. Anda tidak dapat menaikkan batas platform dari kode aplikasi.

Untuk tetap berada dalam batas, pindahkan status massal dari header dan ke badan permintaan, potong cookie, dan ciutkan proxy redundan dan header penelusuran sebelum permintaan mencapai titik akhir. Jika CloudFront distribusi berada di depan titik akhir, teruskan hanya header yang dibutuhkan aplikasi Anda dalam kebijakan permintaan asal.

Jenis kesalahan Fungsi Web

Ketika permintaan ke titik akhir Fungsi Web gagal, Lambda mengembalikan header respons `lambda-web-error-type` yang mengidentifikasi apa yang salah. Nilai menggunakan format`<Prefix>.<ErrorReason>`: awalan memberitahu Anda di mana kesalahan terjadi, dan alasannya menentukan penyebabnya.

Awalan	Dimana kesalahan terjadi
`Platform.`	Lapisan platform Lambda, yang merutekan dan memvalidasi permintaan, memeriksa izin, dan memberlakukan batasan.
`Runtime.`	Runtime terkelola yang memuat dan menjalankan fungsi Anda.
`Application.`	Kode fungsi Anda.

Misalnya, `Runtime.RequestTimeout` berarti runtime terkelola melaporkan bahwa fungsi Anda tidak merespons dalam batas waktu yang dikonfigurasi, dan `Application.InitCodeError` berarti kode inisialisasi Anda sendiri menimbulkan kesalahan.

Lambda menyertakan `lambda-web-error-type` header ketika kesalahan disebabkan oleh permintaan itu sendiri atau oleh fungsi Anda, konfigurasinya, atau kodenya. Lambda menghilangkan header untuk kesalahan layanan internal dan untuk kegagalan otorisasi, sehingga tidak ada detail yang diungkapkan kepada penelepon yang tidak sah. Nilai menunjukkan masalah di jalur antara platform Lambda dan fungsi Anda. Pada jalur permintaan, kesalahan mengidentifikasi di mana permintaan gagal. Di jalur respons, gunakan nilai bersama dengan telemetri aplikasi Anda untuk memahami apa yang terjadi.

Jenis kesalahan	Deskripsi
`Platform.MalformedRequest`	Lambda tidak dapat mengurai permintaan, atau baris permintaan atau metode tidak diterima.
`Platform.MissingHostHeader`	Permintaan tidak menyertakan header Host, jadi Lambda tidak dapat menyelesaikan titik akhir target.
`Platform.InvalidEndpoint`	Header Host tidak menyelesaikan ke titik akhir yang valid dan sudah ada.
`Platform.HeaderTooLarge`	Bagian header permintaan melebihi batas ukuran header Lambda.
`Platform.DisallowedHeader`	Permintaan berisi header yang tidak diterima Lambda. Lambda mengembalikan HTTP 400.
`Platform.UnsupportedHttpVersion`	Permintaan menggunakan versi HTTP selain HTTP/1.1, yang memerlukan Fungsi Web.
`Platform.CannotAssumeRole`	Lambda tidak dapat mengambil peran eksekusi fungsi Anda.
`Platform.KMSDecryptionError`	Lambda tidak dapat menggunakan AWS KMS kunci yang dikonfigurasi untuk mendekripsi variabel lingkungan Anda atau data lain yang diperlukan untuk menginisialisasi fungsi Anda.
`Platform.InvalidNetworkConfiguration`	Sumber daya jaringan VPC yang bergantung pada fungsi Anda tidak valid. Untuk menemukan penyebabnya, lihatthe section called “Masalah jaringan dan konektivitas”.
`Platform.RevisionNotActive`	Revisi fungsi yang ditargetkan oleh permintaan tidak dalam status Aktif. Revisi mungkin masih aktif atau mungkin gagal diaktifkan. Periksa status revisi; jika aktivasi masih berlangsung, coba lagi setelah penundaan singkat.
`Platform.EndpointNotActive`	Titik akhir tidak dalam status Aktif. Titik akhir mungkin masih disediakan atau mungkin tidak tersedia. Periksa status titik akhir; jika masih dibuat, coba lagi setelah penundaan singkat.

Jenis kesalahan	Deskripsi
`Platform.EndpointDisabled`	Titik akhir `rateLimit` adalah 0, jadi Lambda menolak semua permintaan dengan HTTP 503. Setel `rateLimit` ke nilai bukan nol untuk melanjutkan lalu lintas.
`Platform.AccountDisabled`	Pemanggilan Fungsi Web dinonaktifkan untuk akun. Lambda mengembalikan HTTP 403. Kontak AWS Dukungan.
`Platform.AccountRequestThrottle`	Kuota tingkat permintaan tingkat akun terlampaui.
`Platform.EndpointRequestThrottle`	Kuota tingkat permintaan titik akhir terlampaui.
`Platform.AccountActiveVCPUArm64Throttle`	Kuota vCPU aktif akun tercapai.
`Platform.EndpointMaxEnvThrottle`	Batas lingkungan maksimum titik akhir tercapai.
`Platform.EndpointConcurrentInitThrottle`	Terlalu banyak lingkungan eksekusi yang diinisialisasi pada saat yang sama untuk titik akhir. Coba lagi dengan backoff eksponensial.
`Runtime.RequestTimeout`	Fungsi Anda tidak mengembalikan respons dalam batas waktu yang dikonfigurasi.
`Runtime.OutOfMemory`	Fungsi Anda melebihi batas memori dan proses dihentikan.
`Runtime.Unreachable`	Proses runtime macet, keluar, atau menolak koneksi.
`Runtime.Exit`	Proses runtime keluar di antara permintaan.
`Runtime.ExitError`	Proses runtime keluar dengan kesalahan.
`Runtime.InvalidResponse`	Header respons runtime melebihi buffer baca.

Jenis kesalahan	Deskripsi
`Runtime.OriginRequestError`	Proses aplikasi gagal saat menangani permintaan. Periksa log aplikasi untuk detailnya.
`Runtime.ImportCodeError`	Runtime tidak dapat mengimpor atau menyelesaikan titik masuk Anda, misalnya modul yang hilang atau jalur yang buruk.
`Runtime.InitTimeout`	Fungsi Anda melebihi batas waktu init saat menyediakan kotak pasir.
`Application.InitCodeError`	Kode inisialisasi Anda menimbulkan kesalahan sebelum fungsi mulai melayani permintaan.
`Application.TimeoutListenError`	Kode inisialisasi Anda tidak mulai mendengarkan pada port sebelum batas waktu inisialisasi.

Untuk metrik pelambatan yang mengidentifikasi batas mana yang diterapkan (untuk jenis`Platform.AccountRequestThrottle`,, `Platform.EndpointRequestThrottlePlatform.AccountActiveVCPUArm64Throttle`, dan `Platform.EndpointMaxEnvThrottle` kesalahan), lihat. the section called “Penggunaan regional dan metrik pelambatan”

Log atau metrik tidak ada

- Verifikasi bahwa peran eksekusi dapat membuat grup log dan aliran dan menempatkan peristiwa log untuk grup log tujuan dan `.local` pendampingnya.
- Lihat di grup log terpusat di Wilayah asal. Jangan membuat diagnostik yang bergantung pada Replica-region atau `.local` grup log sementara.
- Atur `systemLogLevel` ke `DEBUG` saat Anda membutuhkannya`platform.requestComplete`.
- Izinkan penundaan pengiriman telemetri setelah membuat atau memperbarui sumber daya. Jangan memperlakukan ketidakhadiran sementara peristiwa metrik atau log sebagai bukti bahwa permintaan tidak berjalan.
- Ingatlah bahwa metrik jarang. Konfigurasikan alarm untuk memperlakukan metrik kesalahan yang hilang sebagai tidak melanggar jika perlu.

Jika tidak ada log aplikasi yang muncul sama sekali, peran eksekusi mungkin tidak memiliki izin Amazon CloudWatch Logs. Lampirkan kebijakan `AWSLambdaBasicExecutionRole` terkelola ke peran eksekusi, atau hibah`logs:CreateLogGroup`,`logs:CreateLogStream`, dan`logs:PutLogEvents`. Untuk izin yang diperlukan, lihat the section called “Peran eksekusi”.

Masalah penyimpanan dan enkripsi kode

Versi objek Amazon S3 yang hilang atau tidak dapat diakses

Konfirmasikan bahwa bucket masih mengaktifkan versi, versi objek yang tepat ada, dan pemanggil dan kepala layanan Lambda memiliki izin baca yang diperlukan. Jika revisi aktif kehilangan akses ke versi objek yang disematkan, buat revisi baru yang mereferensikan versi yang tersedia. Lihat perinciannya di the section called “Kode di Amazon S3”.

AWS KMS kegagalan akses

Periksa izin pemanggil, kebijakan kunci, konteks enkripsi, status kunci, dan izin kepala layanan Lambda. `kms:Decrypt` Untuk `MultiRegion` dan `PerRegion` titik akhir, verifikasi bahwa replika AWS KMS kunci Multi-wilayah ada dan dapat digunakan di setiap Wilayah target. Lihat perinciannya di the section called “Enkripsi saat diam”.

Masalah dengan **`MultiRegion`**dan titik **`PerRegion`**akhir

Gunakan `regionalEndpoints` bidang untuk mengidentifikasi Wilayah yang terpengaruh. Bandingkan `stateReason` dan `updateStatusReason` dengan Daerah yang sehat. Periksa kuota Regional, replika AWS KMS kunci, ketersediaan runtime, dan status replikasi revisi. Setelah memperbaiki penyebabnya, ulangi pembaruan titik akhir. Untuk model replikasi, lihatthe section called “Multi-Region penyebaran”.

Untuk `MultiRegion` titik akhir, perutean berbasis latensi (yang mengarahkan permintaan ke Wilayah dengan latensi terukur terendah) tidak menjamin bahwa setiap permintaan masuk ke Wilayah latensi terendah. Uji dari lokasi klien yang terpengaruh dan periksa log aplikasi untuk menentukan Wilayah mana yang menangani permintaan. Untuk `PerRegion` titik akhir, verifikasi juga DNS eksternal atau konfigurasi perutean pintu depan Anda.

Masalah jaringan dan konektivitas

Sebagian besar masalah konektivitas berasal dari revisi penyajian yang tidak memiliki referensi konektor atau referensi versi konektor yang lebih lama, atau dari subnet konektor, grup keamanan, atau rute yang memblokir lalu lintas.

Gunakan urutan berikut untuk mengisolasi masalah konektivitas:

1. Jalankan `get-web-function-endpoint`. Periksa mode penerapan otomatis dan konfirmasikan revisi atau revisi mana yang menerima lalu lintas.
2. Jalankan `get-web-function-revision` untuk setiap revisi penyajian. Konfirmasikan bahwa statusnya `Active` dan itu merujuk pada versi konektor yang diharapkan.
3. Jalankan `aws lambda-core get-network-connector --identifier` dengan konektor ARN dari revisi. `egressNetworkConnectorArn` Konfirmasikan `State` itu `ACTIVE` dan, jika konektor telah diperbarui, `LastUpdateStatus` itu`Successful`. Jika tidak, gunakan `StateReasonCode` atau `LastUpdateStatusReasonCode` untuk mengidentifikasi penyebabnya, seperti`InsufficientRolePermissions`,`InvalidSubnet`, atau`SubnetOutOfIPAddresses`.
4. Uji tujuan dengan alamat IP untuk memisahkan jangkauan jaringan dari resolusi DNS.
5. Periksa rute subnet konektor, grup keamanan tujuan, dan ACL jaringan.
6. Tinjau Log Aliran VPC pada subnet konektor.

Gejala umum termasuk yang berikut ini.

Gejala	Memeriksa
Internet publik berfungsi, tetapi sumber daya VPC tidak	Revisi penyajian tidak memiliki referensi konektor.
Akses VPC berfungsi untuk beberapa permintaan	Titik akhir merutekan ke dua revisi dengan konfigurasi konektor yang berbeda.
Pembaruan konektor tidak berpengaruh	Revisi penyajian mereferensikan versi konektor yang lebih lama.
Akses publik berhenti setelah menambahkan konektor	Subnet konektor tidak memiliki rute NAT atau IPv6 yang diperlukan.

Gejala	Memeriksa
Satu waktu tujuan VPC habis	Grup keamanan tujuan, tabel rute, atau ACL jaringan memblokir koneksi.
Flow Logs tidak berisi catatan yang cocok	Lalu lintas tidak masuk ke VPC. Periksa kembali revisi penyajian dan referensi konektor.
Nama pribadi gagal tetapi koneksi IP langsung berfungsi	Periksa atribut DNS VPC dan asosiasi zona host pribadi.
IPv4 berfungsi tetapi IPv6 gagal	Periksa versi `DualStack` konektor, rute IPv6, dan aturan keamanan IPv6.
Revisi penyajian adalah `Active` tetapi fungsinya tidak dapat mencapai sumber daya VPC pribadi	Revisi penyajian tidak menetapkan konektor jaringan keluar (). `egressNetworkConnectorArn` Baca bidang dengan`get-web-function-revision` ; jika kosong, buat revisi yang menetapkan`egressNetworkConnectorArn` . Lihat perinciannya di the section called “Jaringan”.

Tidak ada akses interaktif ke lingkungan eksekusi Fungsi Web. Gunakan log aplikasi, header respons titik akhir, bidang status sumber daya, CloudWatch metrik, dan Log Aliran VPC untuk mendiagnosis kegagalan.

Siapkan bundel eskalasi

Jika pemeriksaan sebelumnya tidak mengidentifikasi penyebabnya, kumpulkan informasi berikut sebelum membuka kasus. AWS Dukungan Hal ini memungkinkan insinyur dukungan menghubungkan operasi pesawat kontrol, penyebaran Regional, permintaan, dan telemetri:

- Akun AWS ID, Wilayah rumah, nama fungsi, ID revisi, nama titik akhir, tipe titik akhir, dan Wilayah target.
- Timestamp UTC dan jalur permintaan untuk kegagalan representatif, bersama dengan status HTTP,, `lambda-web-request-id`lambda-web-logstream, dan `lambda-web-error-type` header respons.

- Fungsi, revisi, endpoint, dan Regional`state`, `stateReason`, `updateStatus`, dan `updateStatusReason` nilai-nilai.
- Aplikasi terpusat yang relevan dan peristiwa log sistem, metrik throttle dan kapasitas, dan perkiraan tingkat lalu lintas saat masalah terjadi.
- AWS CLI Versi dan salinan permintaan manajemen yang disunting. Hapus kredensyal, rahasia aplikasi, token, dan data permintaan sensitif.

Informasi Terkait

- the section called “Jaringan”
- the section called “Memantau”

AWS Lambda MicroVM

Jika Anda membangun aplikasi di mana beberapa pengguna atau agen AI terhubung ke lingkungan komputasi dan menjalankan kode (misalnya, lingkungan pengembangan interaktif, CI/CD sistem, atau kotak pasir untuk AI) - Anda memerlukan lingkungan komputasi yang dimulai dengan cepat, tetap terisolasi antar penyewa, dan tidak mengharuskan Anda mengelola server atau jaringan.

AWS Lambda MicroVM dibuat khusus untuk kasus penggunaan ini. Mereka adalah lingkungan komputasi tanpa server yang menyediakan VM-level isolasi dengan kemampuan sistem operasi penuh (misalnya, menginstal paket sistem atau memasang sistem file), kecepatan startup cepat berbasis snapshot, dan kontrol halus atas jaringan masuk (akses port, dukungan untuk HTTP/2, gRPC,) dan jaringan keluar (akses internet publik dan akses VPC WebSockets).

Untuk kasus penggunaan interaktif seperti lingkungan pengembangan interaktif dan kotak pasir AI, lingkungan dapat mengalami periode kemalasan yang lama ketika konteks pengguna akhir beralih ke tugas lain atau saat menunggu AI. MicroVM dapat ditangguhkan saat idle, menjaga memori dan status disk sambil mengurangi biaya. Ketika lalu lintas tiba, mereka dilanjutkan. Anda dapat mengaktifkan suspend-resume otomatis atau terprogram untuk menurunkan biaya idle, sambil mempertahankan kesiapan hampir instan saat pengguna akhir Anda kembali.

AWS Lambda MicroVM memberikan kemampuan inti ini melalui virtualisasi Firecracker, yang mendukung 15 triliun+panggilan bulanan Lambda Functions.

Untuk memulai, lihat Buat MicroVM pertama Anda. Untuk informasi harga, lihat Harga AWS Lambda. Untuk keamanan dan izin, lihatthe section called “Keamanan”. Untuk pengamatan, lihatthe section called “Memantau”. Untuk panduan operasional, lihatthe section called “Praktik terbaik”.

Cara kerja Lambda MicroVMS

1. Anda mengemas kode aplikasi Anda dan a `Dockerfile` ke dalam arsip zip dan mengunggahnya ke Amazon S3.
2. Anda memanggil API Lambda untuk membuat image microVM. Lambda mengeksekusi`Dockerfile`, memulai aplikasi Anda, dan menangkap snapshot dari lingkungan yang diinisialisasi sepenuhnya.
3. Ketika aplikasi Anda membutuhkan lingkungan yang terisolasi — untuk sesi pengguna, pekerjaan, atau kotak pasir — Anda meneleponrun-microvm. Lambda meluncurkan microVM dari snapshot dengan waktu startup yang cepat.

4. Klien terhubung ke microVM yang sedang berjalan melalui titik akhir HTTPS khusus. Tidak diperlukan penyeimbang beban atau infrastruktur masuk.
5. Saat idle, microVM menangguhkan, menjaga memori dan status disk sambil mengurangi biaya. Ini dilanjutkan dengan memori dan status disk utuh saat lalu lintas kembali. Suspend-resume perilaku dapat dikonfigurasi secara otomatis melalui kebijakan siklus hidup atau dipicu langsung melalui pemanggilan API ke `suspend-microvm` dan. `resume-microvm`
6. Ketika sesi berakhir, Anda menghentikan microVM untuk melepaskan semua sumber daya.

Kapan menggunakan Lambda MicroVMS

Lambda MicroVMS sangat cocok untuk kasus penggunaan yang mengeksekusi pengguna atau AI-generated kode dan memerlukan lingkungan eksekusi yang menawarkan isolasi yang kuat, latensi peluncuran dan resume yang cepat, dan kontrol pengembang atas siklus hidup lingkungan dan retensi status. Kasus penggunaan umum meliputi:

- Lingkungan kode interaktif — Lingkungan pengembangan tempat pengguna menulis dan mengeksekusi kode secara real time.
- Kotak pasir eksekusi kode AI — Kotak pasir sementara untuk mengeksekusi kode dengan aman. AI-generated
- Aplikasi analisis data — Notebook Jupyter dan beban kerja pemrosesan data sementara yang mengeksekusi skrip yang disediakan pengguna.
- Pemindaian keamanan — Alat penilaian kerentanan yang membutuhkan lingkungan eksekusi terisolasi.
- Lingkungan pembelajaran penguatan — Kotak pasir terisolasi untuk evaluasi dan pelatihan agen AI, di mana lingkungan baru diputar untuk setiap proses.
- Multi-tenant CI/CD— Pelaksana tugas yang membutuhkan isolasi antar penyewa.
- Server game — Lingkungan hosting yang mengeksekusi skrip yang disediakan pengguna dengan isolasi yang kuat.

Fitur utama

- Startup cepat — MicroVM melanjutkan dari snapshot pra-inisialisasi, melewatkan inisialisasi aplikasi. Lihat Image MicroVM.

- Kontrol siklus hidup — Menangguhkan, melanjutkan, dan hentikan microVM secara terprogram atau otomatis melalui kebijakan idle yang dapat dikonfigurasi. Lihat Menjalankan MicroVMS.
- Jaringan fleksibel - MicroVM mendukung lalu lintas HTTPS masuk pada port yang dapat dikonfigurasi, dengan otentikasi JWE yang disediakan layanan. Akses keluar dapat dikonfigurasi. Akses internet publik atau VPC Anda. Untuk detailnya, lihat babak diJaringan.
- Alokasi sumber daya yang fleksibel — Menyediakan setiap microVM untuk penggunaan baseline atau rata-rata, dengan fleksibilitas untuk menskalakan secara vertikal hingga 4x dari baseline yang dikonfigurasi selama aktivitas puncak. Bayar tarif dasar saat microVM Anda berjalan dan hanya membayar untuk penggunaan aktif di atas baseline.

AWS Lambda Konsep inti MicroVMS

AWS Lambda MicroVMS menggunakan beberapa jenis sumber daya yang Anda buat dan kelola. Halaman ini menjelaskan setiap jenis sumber daya, bagaimana Lambda membangun image microVM Anda menjadi snapshot, dan siklus hidup menyatakan transisi microVM saat runtime — dasar untuk membangun aplikasi dengan microVM.

Konsep Utama

MicroVM

MicroVM adalah sumber daya yang mewakili lingkungan komputasi terisolasi untuk satu penyewa, sesi pengguna, atau pekerjaan. Setiap MicroVM menjalankan sistem operasi Amazon Linux 2023 dengan kemampuan sistem operasi, menyediakan peluncuran dan peluncuran hampir instan. MicroVM menerima permintaan melalui koneksi HTTPS masuk dan dapat ditangguhkan saat idle, menjaga memori dan status disk. MicroVM yang ditangguhkan dilanjutkan saat lalu lintas kembali.

Gambar MicroVM

Gambar MicroVM adalah sumber daya yang mendefinisikan lingkungan aplikasi microVM. Saat Anda membuat gambar, Lambda membuatnya menjadi snapshot yang memungkinkan peluncuran hampir instan (lihat Bagaimana Lambda membangun citra Anda di bawah).

Untuk membuat gambar microVM, Anda menyediakan paket zip yang berisi artefak `Dockerfile` dan aplikasi Anda, diunggah ke Amazon S3. Anda harus menggunakan gambar Lambda-published dasar terkelola sebagai fondasi - tentukan dengan `base-image-arn` parameter. Anda `Dockerfile` menentukan lapisan aplikasi yang dibangun Lambda di atas basis terkelola.

Gambar MicroVM diberi versi. Setiap versi mewakili satu build yang dihasilkan dari artefak kode tertentu dan gambar dasar. Versi berkembang melalui status build (`PENDING`→ `IN_PROGRESS` → `SUCCESSFUL` atau`FAILED`), dan versi yang berhasil dapat disetel ke `ACTIVE` atau`INACTIVE`. Untuk detail tentang status gambar dan manajemen, lihatImage MicroVM.

Konektor jaringan

Konektor jaringan adalah sumber daya yang mengontrol bagaimana lalu lintas mencapai microVM Anda dan bagaimana microVM Anda mencapai layanan eksternal. Anda mengaitkan konektor dengan microVM pada waktu berjalan untuk mengonfigurasi akses masuk dan keluar secara independen.

Build-time dan konektor run-time dapat berbeda, memungkinkan microVM Anda menjangkau lingkungan yang berbeda selama pembuatan gambar versus runtime.

Gunakan Lambda-provided default untuk akses port masuk (dengan otentikasi JWE), akses shell, dan keluar internet publik. Buat konektor jaringan Anda sendiri untuk merutekan lalu lintas keluar melalui VPC Anda.

Bagaimana Lambda membangun citra Anda

Saat Anda membuat atau memperbarui gambar microVM, Lambda melakukan proses pembuatan yang menghasilkan snapshot Firecracker. Snapshot ini menangkap status aplikasi Anda yang diinisialisasi sepenuhnya, memungkinkan peluncuran dan dilanjutkan hampir instan untuk microVM yang berjalan darinya.

Proses pembangunan:

1. Lambda menyediakan microVM baru menggunakan image dasar terkelola yang Anda tentukan.
2. Lambda mengeksekusi `Dockerfile` instruksi Anda untuk menginstal dependensi dan mengkonfigurasi lingkungan Anda.
3. Lambda memulai aplikasi Anda menggunakan `CMD` perintah `ENTRYPOINT` or.
4. Jika Anda mengaktifkan `/ready` hook, Lambda menunggu aplikasi Anda memberi sinyal kesiapan (HTTP 200).
5. Lambda menangkap snapshot dari disk dan status memori, termasuk semua proses yang berjalan.

Saat Anda menjalankan microVM, Lambda mengembalikannya dari snapshot ini. Aplikasi Anda dilanjutkan dari status pra-inisialisasi tanpa mengulangi startup.

Jika aplikasi Anda menghasilkan konten unik selama pembuatan (seperti ID unik, rahasia, atau koneksi jaringan), konten tersebut dibagikan di semua MicroVM yang dijalankan dari versi gambar yang sama. Untuk menghindari hal ini, buat konten unik setelah microVM mulai menggunakan hook `/run` siklus hidup. Untuk detailnya, lihat bagian kompatibilitas snapshot diImage MicroVM.

Siklus hidup MicroVM

Saat runtime, transisi microVM melalui tahapan berikut:

1. Lari - Anda meneleponrun-microvm. Lambda mengembalikan microVM dari snapshot gambar, menetapkan ID unik, dan membuat titik akhir. Transisi MicroVM dari PENDING keRUNNING.
2. Menjalankan — Aplikasi Anda menerima dan memproses permintaan melalui URL titik puncaknya.
3. Men angguhkan - Setelah periode idle yang dapat dikonfigurasi (atau melalui suspend-microvm API), microVM beralih ke. SUSPENDING SUSPENDED Memori dan status disk dipertahankan.
4. Lanjutkan — Transisi microVM dari SUSPENDED langsung kembali ke RUNNING saat lalu lintas tiba (jikaautoResumeEnabled=true) atau Anda menelep resume-microvm on.
5. H entikan — Transisi microVM TERMINATING hingga TERMINATED saat Anda menelepon terminate-microvm atau durasi maksimum terlampaui.

Status

Tabel berikut menjelaskan setiap status microVM. Status ini memungkinkan Anda untuk membangun aplikasi yang andal dan menerapkan penanganan kesalahan yang tepat.

Status	Deskripsi
PENDING	MicroVM sedang disediakan. Sumber daya dialokasikan dan snapshot sedang dimuat.
RUNNING	MicroVM aktif dan menerima lalu lintas melalui URL titik puncaknya. Peng /run ait telah selesai.
SUSPENDING	MicroVM sedang ditangguhkan. Peng /suspend ait sedang dieksekusi. Disk dan memori sedang diperiksa.

Status	Deskripsi
SUSPENDED	MicroVM ditangguhkan. Negara dipertahankan. Tidak ada biaya komputasi yang bertambah. Dapat dilanjutkan atau dihentikan.
TERMINATING	MicroVM sedang dihentikan. Peng /terminate ait sedang dieksekusi. Sumber daya sedang dilepaskan.
TERMINATED	MicroVM telah dihentikan. Ini adalah status terakhir. MicroVM tidak dapat dilanjutkan atau dimulai ulang.

Transisi negara

Tabel berikut menunjukkan transisi yang valid antara status microVM dan apa yang memicu setiap transisi.

Keadaan awal	Status target	Pemicu
PENDING	RUNNING	Penyediaan selesai, /run hook berhasil.
RUNNING	SUSPENDING	Durasi idle terlampaui, atau panggilan suspend-microvm API eksplisit.
SUSPENDING	SUSPENDED	/suspendkait selesai, memori dan status disk diperiksa.
SUSPENDED	RUNNING	Lalu lintas tiba (autoResumeEnabled= true) atau panggilan resume-microvm API eksplisit.
RUNNING	TERMINATING	Panggilan terminate-microvm API eksplisit, atau ter maximumDurationInS econds lampaui.
SUSPENDED	TERMINATING	suspendedDurationSeconds terlampau i, atau panggilan terminate-microvm API eksplisit.

Keadaan awal	Status target	Pemicu
`TERMINATING`	`TERMINATED`	`/terminate` kait selesai, semua sumber daya dilepaskan.

> **Important**
>
> Jika `/run` hook Anda gagal atau habis waktu, microVM dapat beralih langsung ke `TERMINATING` tanpa pernah mencapai`RUNNING`. Terapkan batas waktu dan penanganan kesalahan di kait Anda untuk menghindari kegagalan diam.

Buat Lambda MicroVM pertama Anda

Tutorial ini memandu Anda melalui pembuatan gambar AWS Lambda microVM dan menjalankan microVM darinya. Pada akhirnya, Anda akan memiliki aplikasi yang berjalan yang dapat diakses melalui HTTPS.

Prasyarat

Anda membutuhkan dua hal sebelum memulai:

1. Bucket Amazon S3 di Wil AWS ayah pilihan Anda untuk menyimpan artefak aplikasi Anda (file zip yang akan Anda buat di Langkah 2).
2. Peran build IAM yang diasumsikan Lambda selama pembuatan gambar. Lambda menggunakan peran ini untuk mengunduh artefak kode Anda dari Amazon S3 dan menulis log build ke. CloudWatch

Buat peran IAM dengan kebijakan kepercayaan berikut. Ini memungkinkan layanan Lambda untuk mengambil peran:

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Principal": { "Service": "lambda.amazonaws.com" },
    "Action": ["sts:AssumeRole", "sts:TagSession"]
```

```
  }]
}
```

Lampirkan kebijakan izin berikut ke peran. Ganti <your-bucket-name> dengan nama bucket Amazon S3 Anda:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["s3:GetObject"],
      "Resource": "arn:aws:s3:::<your-bucket-name>/*"
    },
    {
      "Effect": "Allow",
      "Action": ["logs:CreateLogGroup", "logs:CreateLogStream", "logs:PutLogEvents"],
      "Resource": "arn:aws:logs:*:*:*"
    }
  ]
}
```

> **Note**
>
> Jika Anda `Dockerfile` menarik dari repositori AWS ECR pribadi, tambahkan `ecr:GetAuthorizationToken` juga `ecr:BatchGetImage` ke kebijakan izin.

Membuat Image MicroVM Pertama Anda

Gambar MicroVM menangkap aplikasi Anda dalam keadaan terinisialisasi sepenuhnya. Ketika Anda menjalankan microVM dari gambar ini, aplikasi Anda segera dimulai — tidak ada penundaan boot atau inisialisasi.

Langkah 1: Tulis aplikasi Anda dan Dockerfile

Buat server HTTP sederhana yang berjalan di dalam microVM Anda. Contoh ini menggunakan Node.js tanpa dependensi eksternal:

`app.js`

```
// Minimal HTTP server — listens on port 8080
const http = require('http');

const server = http.createServer((req, res) => {
  res.writeHead(200, { 'Content-Type': 'application/json' });
  res.end(JSON.stringify({ status: 'ok', path: req.url }));
});

server.listen(8080, () => {
  console.log('Listening on port 8080');
});
```

Selanjutnya, buat paket `Dockerfile` itu dan mulai aplikasi Anda:

`Dockerfile`

```
# Use a lightweight Node.js runtime for your application layers
FROM node:24-alpine

# Set the working directory inside the container
WORKDIR /app

# Copy your application code
COPY app.js .

# Declare the port your app listens on
EXPOSE 8080

# Start the application — Lambda snapshots the running state
CMD ["node", "app.js"]
```

> **Note**
>
> `FROM`Instruksi menetapkan gambar wadah untuk lapisan aplikasi Anda. Anda dapat menggunakan gambar kontainer yang kompatibel. Gambar dasar yang dikelola Lambda (yang menyediakan sistem operasi MicroVM dan komponen layanan) ditentukan secara terpisah dengan `--base-image-arn` pada Langkah 3.
>
> Jika aplikasi Anda menghasilkan nilai unik (ID, rahasia, atau materi kriptografi), gunakan pustaka generator nomor pseudorandom (CSPRNG) standar bahasa Anda yang aman secara kriptografis untuk memastikan keunikan di seluruh microVM. Jika aplikasi Anda menggunakan OpenSSL, gunakan gambar wadah dasar Lambda yang menyertakan versi

yang kompatibel dengan snapshot. Untuk detailnya, lihat bagian kompatibilitas snapshot di[Image MicroVM](#).

Langkah 2: Kemas dan unggah ke Amazon S3

Kemas `app.js` dan `Dockerfile` ke dalam arsip zip, lalu unggah ke bucket Amazon S3 Anda. Jalankan perintah berikut di terminal Anda:

```
zip app.zip app.js Dockerfile
aws s3 cp app.zip s3://your-bucket-name/app.zip
```

Langkah 3: Buat gambar microVM

Panggil `create-microvm-image` untuk memulai pembangunan. Lambda mengunduh zip Anda dari Amazon S3, menjalankan, memulai aplikasi Anda`Dockerfile`, dan menangkap snapshot Firecracker dari status yang diinisialisasi sepenuhnya:

```
aws lambda-microvms create-microvm-image \
  --name my-first-microvm-image \
  --code-artifact uri=s3://your-bucket-name/app.zip \
  --base-image-arn arn:aws:lambda:us-east-1:aws:microvm-image:al2023-1 \
  --build-role-arn arn:aws:iam::123456789012:role/MicrovmBuildRole
```

Gambar dimulai dalam `CREATING` keadaan. Periksa status build dengan:

```
aws lambda-microvms get-microvm-image \
  --image-identifier my-first-microvm-image
```

Saat build selesai, `state` bidang berubah menjadi`CREATED`:

```
{
  "imageName": "my-first-microvm-image",
  "imageArn": "arn:aws:lambda:us-east-1:123456789012:microvm-image:my-first-microvm-
image",
  "state": "CREATED",
  "imageVersion": "1.0",
  ...
}
```

Jika statusnya adaCREATE_FAILED, periksa log build di CloudWatch bawah/aws/lambda-microvms/my-first-microvm-image.

Menjalankan MicroVM pertama Anda

Setelah image microVM Anda mencapai CREATED status, Anda dapat menjalankan microVMS darinya. Setiap gambar dapat meluncurkan banyak microVM — satu per penyewa, sesi pengguna, atau pekerjaan.

Jalankan microVM dengan perintah berikut:

```
aws lambda-microvms run-microvm \
  --image-identifier my-first-microvm-image \
  --ingress-network-connectors "arn:aws:lambda:us-east-1:aws:network-connector:aws-
network-connector:ALL_INGRESS" \
  --egress-network-connectors "arn:aws:lambda:us-east-1:aws:network-connector:aws-
network-connector:INTERNET_EGRESS" \
  --idle-policy
 '{"autoResumeEnabled":true,"maxIdleDurationSeconds":900,"suspendedDurationSeconds":300}'
```

Parameter dijelaskan:

- `--ingress-network-connectors`- Mengaktifkan lalu lintas HTTPS masuk ke microVM Anda di semua port. Ini adalah Lambda-managed konektor.
- `--egress-network-connectors`- Mengaktifkan akses internet keluar dari microVM Anda. Ini adalah Lambda-managed konektor.
- `--idle-policy`— Mengkonfigurasi perilaku suspend-resume otomatis. Kebijakan ini menangguhkan microVM setelah 15 menit tidak aktif, tetap ditangguhkan hingga 5 menit, dan dilanjutkan secara otomatis saat lalu lintas tiba.

Responsnya mencakup ID microVM dan URL titik akhir:

```
{
  "microvmId": "mvm-01234567-abcd-ef01-2345-6789abcdef01",
  "state": "PENDING",
  "endpoint": "mvm-01234567-abcd-ef01-2345-6789abcdef01.lambda-microvm.us-
east-1.on.aws",
  ...
}
```

Tunggu negara mencapai`RUNNING`:

```
aws lambda-microvms get-microvm \
  --microvm-identifier mvm-01234567-abcd-ef01-2345-6789abcdef01
```

Menghubungkan ke MicroVM Anda

Semua permintaan ke titik akhir microVM memerlukan token otentikasi. Hasilkan satu dengan:

```
aws lambda-microvms create-microvm-auth-token \
  --microvm-identifier mvm-01234567-abcd-ef01-2345-6789abcdef01 \
  --expiration-in-minutes 30 \
  --allowed-ports '[{"allPorts":{}}]'
```

Tanggapan termasuk token di `authToken` lapangan. Gunakan untuk mengirim permintaan ke aplikasi Anda yang sedang berjalan:

```
curl https://mvm-01234567-abcd-ef01-2345-6789abcdef01.lambda-microvm.us-east-1.on.aws/
 \
  -H "X-aws-proxy-auth: <token-value>"
```

Anda akan melihat tanggapan dari aplikasi Anda:

```
{"status":"ok","path":"/"}
```

MicroVM Anda berjalan dan melayani lalu lintas. Aplikasi yang Anda tulis di Langkah 1 aktif di URL titik akhir.

Bersihkan

Untuk menghindari biaya yang sedang berlangsung, hentikan microVM setelah selesai:

```
aws lambda-microvms terminate-microvm \
  --microvm-identifier mvm-01234567-abcd-ef01-2345-6789abcdef01
```

Langkah selanjutnya

- Pelajari tentang Konsep inti memahami proses snapshot, status siklus hidup, dan gambar dasar.
- Jel Image MicroVM ajahi untuk mempelajari tentang kait pembuatan gambar, pembuatan versi, dan kompatibilitas snapshot.

- Lihat Menjalankan MicroVMS contoh SDK, kait siklus hidup, dan strategi penskalaan.
- Lihat the section called “Integrasi” integrasi layanan yang didukung.

Gambar MicroVM

Bagian ini menjelaskan cara membuat, mengkonfigurasi, memperbarui, dan mengelola gambar microVM.

Gambar MicroVM adalah sumber daya yang mendefinisikan sistem file dan lingkungan aplikasi dari microVM. Gambar MicroVM mencakup lingkungan runtime Anda, kode aplikasi, dan program pendukung seperti proses latar belakang dan agen pengamatan. Untuk membuat gambar microVM, Anda menyediakan paket zip yang berisi artefak `Dockerfile` dan aplikasi Anda, yang Anda unggah ke Amazon S3.

Anda `Dockerfile` menentukan bagaimana aplikasi Anda dikemas. Lambda membangun image wadah aplikasi Anda dengan menjalankan `Dockerfile` di atas lingkungan sistem operasi yang disediakan oleh image dasar Lambda-managed MicroVM. Gambar dasar MicroVM dijelaskan di bawah ini, di bagian berjudul —Gambar dasar MicroVM.

Anda dapat memperbarui gambar dasar MicroVM Anda untuk memperbarui kode aplikasi atau konfigurasi untuk microVM Anda. Setiap pembaruan yang Anda picu membuat versi gambar MicroVM baru.

Bagaimana Lambda membangun image microVM

Saat Anda membuat gambar microVM, Lambda:

- Mengambil artefak paket Anda dari Amazon S3.
- Memulai microVM baru dari gambar Lambda-managed dasar.
- Mengeksekusi instruksi di Anda`Dockerfile`.
- Luncurkan aplikasi Anda menggunakan `CMD` instruksi `ENTRYPOINT` atau.
- Menunggu inisialisasi selesai, ditandai oleh kait siklus hidup Anda.
- Menangkap snapshot dari disk dan status memori.

Setelah proses snapshot selesai, image microVM Anda memasuki `CREATED` status. Anda sekarang dapat menggunakan gambar microVM ini untuk membuat microVM, dan setiap gambar microVM dapat digunakan untuk membuat beberapa microVM independen. Sebuah microVM yang dijalankan

dari image microVM dilanjutkan langsung dari status snapshot, memberikan waktu startup yang cepat. Setiap gambar microVM dapat digunakan untuk menjalankan beberapa microVM, hingga batas yang tersedia untuk akun Anda.

Untuk panduan langkah demi langkah mengemas kode Anda dan membuat gambar microVM pertama Anda, lihat. Buat MicroVM pertama Anda

Ukuran MicroVM

Lambda MicroVMS menggunakan model baseline-peak yang menghilangkan kebutuhan untuk menyesuaikan ukuran setiap lingkungan komputasi untuk aktivitas puncak. Anda mengonfigurasi sumber daya komputasi dasar untuk microVM Anda. Selama aktivitas puncak, microVM Anda dapat secara otomatis menskalakan secara vertikal hingga 4x baseline. Anda membayar tarif dasar saat microVM Anda berjalan dan hanya membayar untuk apa yang Anda gunakan secara aktif di atas baseline, ditagih per detik.

Anda mengatur baseline dengan menggunakan `memory` parameter saat membuat image microVM Anda. vCPU berskala secara proporsional dengan memori (2 GB = 1 vCPU). Baseline default adalah 2 GB/1 vCPU.

Tabel berikut mencantumkan ukuran yang tersedia:

Baseline	Puncak	Ruang disk maks
Memori 0,5 GB, 0,25 vCPU	Memori 2 GB, 1 vCPU	8GB
Memori 1 GB, 0,5 vCPU	Memori 4 GB, 2 vCPU	8GB
Memori 2 GB, 1 vCPU (default)	Memori 8 GB, 4 vCPU	8GB
Memori 4 GB, 2 vCPU	Memori 16 GB, 8 vCPU	16GB
Memori 8 GB, 4 vCPU	Memori 32 GB, 16 vCPU	32 GB

Gambar dasar MicroVM

Sebuah image dasar MicroVM berfungsi sebagai dasar untuk image microVM Anda. Lambda menerbitkan gambar dasar MicroVM yang menyediakan sistem operasi Amazon Linux 2023

dan komponen layanan yang diperlukan untuk menjalankan MicroVM. Saat Anda membuat atau memperbarui image microVM, Lambda memulai microVM baru dari image dasar ini dan menjalankan `Dockerfile` instruksi Anda dalam lingkungan sistem operasi ini.

Lambda secara berkala merilis versi baru dari image dasar MicroVM yang dikelola layanan, seperti saat menerapkan patch keamanan, untuk memperbarui sistem operasi atau komponen layanan. Secara default, versi terbaru dari image dasar yang dikelola layanan berlaku ketika Anda adalah image microVM creating/updating Anda sendiri. Untuk pemecahan masalah atau debugging, Anda dapat mengganti versi gambar berbasis layanan yang dikelola secara opsional saat membuat gambar microVM Anda sendiri menggunakan parameter. `base-image-version`

Versi gambar dasar mengikuti siklus hidup penghentian:

- **`AVAILABLE`**— Saat ini, direkomendasikan untuk digunakan.
- **`DEPRECATED`**(60 hari) - Ada versi yang lebih baru. Anda masih bisa membangun dan menjalankan.
- **`EXPIRING`**(30 hari) - Tidak dapat membuat gambar baru. Gambar yang ada masih bisa berjalan.
- **`EXPIRED`**— Tidak dapat membangun atau menjalankan. Bangun kembali gambar Anda pada versi yang didukung.
- **`RECALLED`**— Segera tidak tersedia karena masalah keamanan kritis (jarang).

Agar tetap terkini, pantau pemberitahuan penghentian dan bangun kembali gambar microVM Anda saat versi gambar dasar baru dirilis.

Perhatikan bahwa image dasar MicroVM berbeda dari image basis kontainer yang Anda tentukan di Dockerfiles Anda. Sementara yang pertama mendefinisikan lingkungan sistem operasi untuk microVM Anda, yang terakhir menentukan image wadah dasar mana yang akan digunakan saat mengemas aplikasi Anda untuk digunakan dengan Lambda MicroVMS. Lihat bagian di Gambar basis kontainer untuk lebih detailnya.

Gunakan API berikut untuk menemukan image dasar MicroVM terkelola yang tersedia dan versinya:

```
# List all managed MicroVM base images
aws lambda-microvms list-managed-microvm-images

# List the versions of a specific managed MicroVM base image
aws lambda-microvms list-managed-microvm-image-versions \
  --image-identifier arn:aws:lambda:us-east-1:aws:microvm-image:al2023-1
```

Kait pembuatan gambar MicroVM

Lambda menyediakan kait pembuatan gambar microVM yang memungkinkan Anda memverifikasi kebenaran aplikasi dan mengoptimalkan kinerja selama pembuatan gambar microVM. Hooks berjalan sebelum Lambda mengambil snapshot yang digunakan untuk menginisialisasi setiap microVM. Setiap hook adalah titik akhir HTTP yang diekspos aplikasi Anda dan yang dipanggil Lambda selama pembuatan. Dengan menanggapi permintaan ini, Anda mengontrol dan memvalidasi proses pembuatan gambar microVM. Lambda menggunakan kode Status HTTP untuk menentukan apakah hook berhasil diselesaikan.

> **Important**
>
> Jika Anda mengonfigurasi kait apa pun, Anda harus menentukan port yang didengarkan aplikasi Anda untuk permintaan kait.

Kait	Jalur	Rincian	Kode status HTTP	Batas waktu
/siap	`/aws/lambda-microvms/runtime/v1/ready`	Dipanggil selama pembuatan gambar microVM, setelah aplikasi Anda dimulai melalui `ENTRYPOINT` atau`CMD`. Menandakan bahwa aplikasi Anda siap untuk diambil.	HTTP 503: Belum siap; Lambda mencoba lagi sampai batas waktu. HTTP 200: Inisialisasi selesai; Lambda mengambil snapshot.	1—3600 detik () `readyTimeoutInSeconds`
/memvalidasi	`/aws/lambda-microvms/runtime/v1/validate`	Dipanggil setelah build selesai, pada microVM baru dimulai dari gambar yang dibuat. Mengonfirmasi aplikasi berfungsi dengan benar saat dilanjutkan.	HTTP 503: Validasi membutuhkan lebih banyak waktu untuk diselesaikan; Lambda mencoba lagi sampai batas waktu. HTTP 200: Validasi lulus.	1—3600 detik () `validateTimeoutInSeconds`

> **Important**
>
> Saat mengembalikan HTTP 503, kembalikan segera daripada menahan permintaan terbuka saat Anda menunggu. Jika batas waktu berlalu saat permintaan ditahan terbuka, Lambda mengakhiri build.

> **Note**
>
> Anda juga dapat menggunakan hook /validate untuk mengoptimalkan waktu startup. Untuk melakukannya, jalankan muatan tiruan selama validasi. Ini memungkinkan Lambda untuk melacak wilayah yang diakses dari snapshot Anda dan mengoptimalkan pengambilannya selama startup MicroVM.

Memperbarui gambar microVM

Anda dapat memperbarui image microVM yang ada dengan memanggil `update-microvm-image` API. Setiap pembaruan memicu pembuatan versi gambar microVM baru. Anda biasanya memperbarui image microVM ke:

- Menyebarkan kode aplikasi baru — Arahkan ke artefak kode baru (zip baru yang diunggah ke Amazon S3) untuk mengirimkan versi baru aplikasi Anda.
- Pindah ke image dasar MicroVM yang lebih baru — Ubah gambar dasar MicroVM ARN untuk meningkatkan ke versi yang lebih baru dari image dasar MicroVM Lambda. Untuk informasi lebih lanjut, lihat Penambalan gambar MicroVM danGambar dasar MicroVM.
- Ubah peran build — Perbarui peran build ARN saat izin yang dibutuhkan Lambda selama perubahan build, seperti saat artefak kode Anda pindah ke bucket Amazon S3 yang berbeda atau Anda mulai menarik dari repositori ECR pribadi.
- Sesuaikan konfigurasi runtime — Ubah kait, variabel lingkungan, atau kemampuan untuk mengkonfigurasi ulang bagaimana image microVM Anda dibuat dan dijalankan.
- Perbarui deskripsi - Ubah deskripsi gambar MicroVM untuk merekam apa yang berubah dalam versi ini.

Perintah CLI berikut menunjukkan bagaimana Anda dapat memperbarui gambar microVM. `--build-role-arn`Param `--base-image-arn` eter dan diperlukan pada setiap `update-microvm-`

`image` panggilan yang memicu build baru, bahkan ketika Anda hanya mengubah artefak kode — menghilangkannya menghasilkan: `ValidationException`

```
aws lambda-microvms update-microvm-image \
  --image-identifier arn:aws:lambda:us-east-1:123456789012:microvm-image:my-microvm-image \
  --code-artifact uri=s3://my-bucket/deployments/app-v2.zip \
  --base-image-arn arn:aws:lambda:us-east-1:aws:microvm-image:al2023-1 \
  --build-role-arn arn:aws:iam::123456789012:role/MicrovmBuildRole \
  --description "Updated with v2 application code"
```

Status gambar dan membangun status

Setiap kali Anda membuat atau memperbarui gambar microVM, Lambda menghasilkan versi baru yang dibangun dari artefak kode dan gambar dasar Anda. Sebuah image microVM dapat memiliki banyak versi dari waktu ke waktu, dan Anda menjalankan MicroVMS dari versi tertentu.

Tiga negara independen melacak aspek yang berbeda dari siklus hidup:

- Status gambar — siklus hidup keseluruhan sumber daya gambar microVM (sedang dibuat, siap digunakan, diperbarui, gagal, atau dihapus).
- Status versi — kemajuan pembuatan versi tertentu (tertunda, membangun, berhasil, atau gagal). Periksa `stateReason` atau CloudWatch log (`/aws/lambda-microvms/<image-name>`) untuk detail kegagalan.
- Aktivasi versi — apakah versi yang berhasil dibuat diizinkan untuk menjalankan MicroVMS. Lambda menetapkan versi baru secara `ACTIVE` otomatis; Anda dapat mengatur versi `INACTIVE` untuk menonaktifkannya tanpa menghapusnya.

Status	Kemungkinan nilai	Transisi dikendalikan oleh
Status gambar	`CREATING`, `CREATED`, `CREATION_FAILED`, `UPDATING`, `UPDATED`, `UPDATE_FAILED`, `DELETING`, `DELETED`, `DELETION_FAILED`	Lambda (otomatis)
Status versi	`PENDING`, `IN_PROGRESS`, `SUCCESSFUL`, `FAILED`	Lambda (otomatis)

Status	Kemungkinan nilai	Transisi dikendalikan oleh
Aktivasi versi	`ACTIVE`, `INACTIVE`	Anda (`update-microvm-image-version --state`)

Untuk menjalankan microVM dari versi, status gambar harus `CREATED` atau`UPDATED`, status versi harus`SUCCESSFUL`, dan versi harus `ACTIVE` demikian.

> **Note**
>
> Negara-negara ini independen. Gambar di `CREATED` negara bagian dapat berisi versi yang statusnya`FAILED`.

```
# De-activate a version
aws lambda-microvms update-microvm-image-version \
  --image-identifier my-image \
  --image-version 1.0 \
  --state INACTIVE
```

Variabel lingkungan

Variabel lingkungan diatur pada waktu pembuatan gambar microVM melalui `environmentVariables` bidang (maksimum 50 variabel). Ini disuntikkan ke dalam wadah selama proses pembuatan snapshot. Anda dapat melewati payload yang diatur secara dinamis saat menjalankan microVM baru. Lihat bagian tentang menjalankan microVM Anda untuk mempelajari lebih lanjut.

Cara menggunakan snapshot

Ketika AWS Lambda MicroVMS membangun gambar MicroVM, ia menangkap snapshot Firecracker dari aplikasi Anda yang diinisialisasi sepenuhnya. Halaman ini menjelaskan apa yang terkandung dalam snapshot, cara menangani keunikan di seluruh microVM yang berbagi snapshot yang sama, dan praktik terbaik untuk aplikasi yang kompatibel dengan snapshot.

Apa yang diambil snapshot

Snapshot Firecracker menangkap status berikut dari microVM Anda pada saat pembuatan gambar selesai:

- Status memori proses yang sedang berjalan — Semua proses yang diluncurkan oleh`ENTRYPOINT`/`CMD`, termasuk daemon latar belakang, pekerjaan cron, dan proses anak.
- Status disk — Sistem file root termasuk semua file yang ditulis selama fase build (paket yang diinstal, binari yang dikompilasi, file konfigurasi).
- Koneksi jaringan dan deskriptor file — Setiap koneksi jaringan yang diinisialisasi, pegangan file, pipa, dan deskriptor lainnya.

Saat Anda menjalankan microVM, Lambda mengembalikannya dari snapshot ini. Aplikasi Anda dilanjutkan persis di tempat pembuatan versi gambar MicroVM ditinggalkan.

Pertimbangan kompatibilitas

Karena semua MicroVM yang diluncurkan dari versi gambar yang sama berbagi status awal yang identik, pertimbangkan hal berikut saat membangun aplikasi dengan MicroVM Lambda:

- Keunikan — Jika kode Anda menghasilkan konten unik selama fase pembuatan versi gambar (ID unik, rahasia, atau entropi untuk pseudorandomness), konten tersebut dibagikan di semua microVM yang dijalankan dari versi gambar yang sama. Hasilkan konten unik setelah microVM baru dimulai, bukan selama pembuatan gambar. Gunakan kait `/run` siklus hidup untuk mengatur ulang konten unik yang dibuat sebelumnya.
- Koneksi jaringan — Koneksi yang dibuat selama fase pembuatan versi gambar mungkin perlu dibuat kembali ketika microVM dijalankan dari versi gambar. Validasi status koneksi dan buat kembali koneksi seperlunya. Dalam kebanyakan kasus, koneksi yang dibuat oleh AWS SDK secara otomatis dibangun kembali.

Menangani keunikan dengan snapshot

Semua microVM yang dijalankan dari gambar yang sama berbagi status awal yang identik. Untuk mempertahankan keunikan di seluruh microVM:

- Hasilkan ID unik, rahasia, dan nilai acak setelah microVM dimulai dan bukan selama pembuatan versi gambar. Gunakan kait `/run` siklus hidup untuk mengatur ulang status unik apa pun.

- Gunakan generator nomor pseudorandom yang aman secara kriptografis (CSPRNGs) untuk bahasa pemrograman Anda. Pastikan pustaka kriptografi standar bahasa Anda menerima entropinya dari perangkat atau perangkat. `/dev/random` `/dev/urandom` Ini adalah perilaku default untuk pustaka kriptografi untuk bahasa pemrograman populer seperti Java 11+ (SecureRandom library), Node.js (crypto.randombytes library), Python 3.12+ (library), dan Dotnet8+ (Secrets.SystemRandomlibrary). Cryptography.RandomNumberGenerator Perangkat lunak yang membaca dari `/dev/random` atau `/dev/urandom` mempertahankan keacakan saat digunakan dengan snapshot gambar microVM.
- Jika kode aplikasi Anda menggunakan OpenSSL, gunakan image wadah dasar AWS yang disediakan dari `public.ecr.aws/lambda/microvms:al2023-minimal` untuk membangun image microVM Anda. Gambar ini berisi OpenSSL versi AWS-patched yang kompatibel dengan snapshotting. Jika Anda ingin menggunakan image dasar Anda sendiri dengan OpenSSL, gunakan versi tambalan OpenSSL (openssl-snapsafe-libs) yang tercantum dalam daftar paket Amazon Linux 2023. https://docs.aws.amazon.com/linux/al2023/release-notes/all-packages.html

Penambalan gambar MicroVM

Ketika ada image dasar microVM baru yang tersedia, Anda dapat mengeluarkan `update-microvm-image` panggilan untuk memicu pembuatan gambar microVM dengan patch terbaru — baik menghilangkan `base-image-version` argumen (untuk terbaru) atau menentukan argumen dengan versi terbaru.

Gambar basis kontainer

Lambda MicroVMS menjalankan aplikasi Anda sebagai wadah dalam lingkungan sistem operasi MicroVM. Anda menentukan wadah itu dengan Anda`Dockerfile`, dan `FROM` instruksi di Anda `Dockerfile` menetapkan gambar dasar wadah untuk aplikasi Anda.

Anda dapat memulai dengan gambar wadah dasar Lambda untuk Amazon Linux 2023 (`public.ecr.aws/lambda/microvms:al2023-minimal`) dan menambahkan `Dockerfile` instruksi Anda di atasnya, atau menggunakan gambar wadah dasar Anda sendiri. Saat menggunakan gambar kontainer Anda sendiri, validasi persyaratan berikut:

Persyaratan

- Gambar basis kontainer harus kompatibel dengan arsitektur CPU target.

- Gambar basis kontainer dari reposit AWS ori ECR pribadi memerlukan peran build yang harus dimiliki `ecr:GetAuthorizationToken` dan `ecr:BatchGetImage` izin.
- Gambar basis kontainer harus didasarkan pada sistem operasi Linux.
- Gambar basis kontainer harus dapat diakses dari infrastruktur build Lambda (internet publik atau repositori ECR di AWS akun yang sama).
- Gambar dasar kontainer harus kompatibel dengan snapshot, lihat instruksi di bawah ini.

Snapshot-compatible gambar dasar

Karena Lambda MicroVMS memulai setiap microVM dari snapshot pra-inisialisasi, gambar dasar harus kompatibel dengan snapshot. Sebaiknya tinjau bagian Pertimbangan kompatibilitas saat menggunakan gambar dasar Anda sendiri dengan Lambda MicroVMS.

Menggunakan gambar ECR pribadi

Referensikan gambar dasar wadah ECR pribadi Anda dalam `FROM` instruksi Anda`Dockerfile`:

```
FROM 123456789012.dkr.ecr.us-east-1.amazonaws.com/my-base:latest
WORKDIR /app
COPY . .
CMD ["./my-app"]
```

Tambahkan izin berikut ke peran build Anda:

```
{
  "Effect": "Allow",
  "Action": [
    "ecr:GetAuthorizationToken",
    "ecr:BatchCheckLayerAvailability",
    "ecr:GetDownloadUrlForLayer",
    "ecr:BatchGetImage"
  ],
  "Resource": "*"
}
```

Kemampuan sistem operasi

Secara default, Lambda MicroVM berjalan dengan seperangkat kemampuan Linux standar. Anda dapat memberikan kemampuan Linux yang ditingkatkan menggunakan

`additionalOsCapabilities` bidang saat membuat atau memperbarui gambar microVM. Satu-satunya nilai yang di-support adalah `["ALL"]`. Kemampuan yang ditingkatkan memungkinkan operasi seperti memasang sistem file, membuat ruang nama jaringan, atau menjalankan program eBPF. Kemampuan diterapkan dalam batas isolasi VM dan tidak mempengaruhi host atau microVM lainnya.

```
aws lambda-microvms create-microvm-image \
  --name my-network-tool \
  --code-artifact uri=s3://my-bucket/app.zip \
  --base-image-arn arn:aws:lambda:us-east-1:aws:microvm-image:al2023-1 \
  --build-role-arn arn:aws:iam::123456789012:role/BuildRole \
  --additional-os-capabilities '["ALL"]'
```

Menjalankan dan menggunakan MicroVMS

Bagian ini menjelaskan cara memulai MicroVM, menyambung ke aplikasi yang sedang berjalan, mengelola siklus hidup MicroVM, dan menangani penskalaan.

Memulai MicroVM

Gunakan `run-microvm` perintah untuk meluncurkan microVM baru dari gambar tertentu. Lambda menyediakan sumber daya yang diperlukan, membuat titik akhir HTTPS khusus, dan memulai aplikasi Anda dari snapshot gambar.

```
aws lambda-microvms run-microvm \
  --image-identifier arn:aws:lambda:us-east-1:123456789012:microvm-image:my-microvm-image \
  --ingress-network-connectors "arn:aws:lambda:us-east-1:aws:network-connector:aws-network-connector:ALL_INGRESS" \
  --egress-network-connectors "arn:aws:lambda:us-east-1:aws:network-connector:aws-network-connector:INTERNET_EGRESS" \
  --idle-policy
 '{"autoResumeEnabled":true,"maxIdleDurationSeconds":900,"suspendedDurationSeconds":1800}'
 \
  --maximum-duration-in-seconds 14400
```

MicroVM dibuat saat Anda menelep `run-microvm` on. Setiap microVM memiliki titik akhir khusus sendiri. Tidak ada penyeimbangan beban di seluruh microVM dari satu titik akhir — setiap titik akhir ditautkan ke microVM tunggal.

Satu-satunya parameter yang diperlukan adalah `--image-identifier` (yang harus menjadi ARN dari gambar microVM). Semua parameter lainnya adalah opsional.

Parameter Kunci

Parameter	Deskripsi
`--image-identifier`	(Diperlukan) ARN dari image microVM untuk dijalankan.
`--image-version`	Versi gambar microVM yang akan dijalankan. Defaultnya ke versi aktif terbaru.
`--execution-role-arn`	Peran IAM yang menyediakan izin runtime untuk microVM untuk berinteraksi dengan layanan lain AWS .
`--idle-policy`	Mengontrol penangguhan otomatis dan melanjutkan perilaku. Lihat konfigurasi kebijakan idle di bagian berikut.
`--maximum-duration-in-seconds`	Umur total maksimum microVM, termasuk waktu berjalan dan ditangguhkan. Lambda mengakhiri microVM ketika durasi ini terlampaui. Rentang: 1—28.800 detik (8 jam).
`--run-hook-payload`	Muatan string (maks 16 KB) dikirimkan ke hook `/run` siklus hidup saat microVM dimulai.
`--logging`	Konfigurasi logging. Sesuaikan grup CloudWatch log dan streaming, atau nonaktifkan pencatatan sepenuhnya.
`--ingress-network-connectors`	ARN konektor masuk yang memungkinkan konektivitas HTTPS masuk.
`--egress-network-connectors`	ARN konektor keluar untuk konektivitas keluar (internet atau VPC).

> **Note**
>
> Untuk menonaktifkan konektivitas masuk, gunakan Lambda-provided `NO_INGRESS` konektor. Untuk detail selengkapnya tentang konektor jaringan, lihatthe section called “Jaringan”.

Konfigurasi kebijakan idle

Saat diaktifkan, kebijakan idle mengontrol penangguhan otomatis dan dimulainya kembali. Kehadiran lalu lintas melalui titik akhir microVM menandakan aktivitas. Jika tidak ada lalu lintas yang tiba untuk durasi idle yang dikonfigurasi, microVM diperlakukan sebagai idle dan ditangguhkan.

Bidang	Deskripsi
`autoResumeEnabled`	Ketika`true`, microVM secara otomatis dilanjutkan ketika lalu lintas tiba di titik puncaknya saat ditangguhkan.
`maxIdleDurationSeconds`	Jumlah detik tanpa lalu lintas setelah itu microVM ditangguhkan. Maksimum: 28.800 (8 jam).
`suspendedDurationSeconds`	Jumlah detik microVM tetap dalam keadaan tersuspensi sebelum Lambda menghentikannya.

> **Note**
>
> Untuk aplikasi asinkron yang tidak secara aktif mengirim atau menerima lalu lintas melalui titik akhir, nonaktifkan penangguhan otomatis atau konfigurasikan durasi idle yang sesuai.

Muatan waktu pelaksanaan

Dengan `runHookPayload` parameter, Anda dapat meneruskan data konfigurasi per-microVM (string maks 16 KB) pada waktu berjalan. Lambda mengirimkan muatan ini sebagai bagian dari badan permintaan ke kait `/run` siklus hidup. Lambda juga menyuntikkan `microvmId` ke badan permintaan.

H `/run` ook menerima badan JSON dengan struktur berikut:

```
{
  "microvmId": "mvm-01234567-abcd-ef01-2345-6789abcdef01",
  "runHookPayload": "tenant-specific-string"
}
```

Gunakan payload runtime untuk menyediakan konfigurasi yang bervariasi per microVM — misalnya, ID penyewa, token sesi, URL yang ditandatangani, atau jalur Manajer Rahasia. Tidak seperti variabel

lingkungan (yang diatur pada tingkat gambar dan dibagikan di semua microVM dari gambar itu), payload hook run unik untuk setiap microVM.

```
aws lambda-microvms run-microvm \
  --image-identifier arn:aws:lambda:us-east-1:123456789012:microvm-image:my-microvm-image \
  --run-hook-payload 'tenant-specific-string'
```

Ketika Anda tidak lagi membutuhkan microVM, hentikan untuk menghentikan semua biaya. Untuk petunjuk, lihat the section called “Mengakhiri MicroVM”.

Menghubungkan ke MicroVM

Setiap microVM mendapatkan URL titik akhir HTTPS publik yang unik, ditetapkan saat Anda menelep `run-microvm` on. Anda terhubung ke aplikasi Anda yang berjalan di dalam microVM melalui URL ini.

Autentikasi

Semua permintaan ke titik akhir microVM memerlukan token otentikasi JWE. Tidak ada opsi akses yang tidak diautentikasi. Hasilkan token dengan`create-microvm-auth-token`:

```
aws lambda-microvms create-microvm-auth-token \
  --microvm-identifier microvm-id \
  --expiration-in-minutes 30 \
  --allowed-ports '[{"allPorts":{}}]'
```

Token dicakup ke port tertentu dan memiliki kedaluwarsa yang dapat dikonfigurasi. Anda dapat membatasi akses ke satu port, rentang port, atau semua port:

```
{ "port": number }
{ "range": { "startPort": N, "endPort": N } }
{ "allPorts": {} }
```

Perutean port

Secara default, Lambda merutekan lalu lintas masuk ke port 8080 dalam microVM Anda. Untuk merutekan ke port yang berbeda, sertakan `X-aws-proxy-port` header dalam permintaan Anda. Port target harus berada dalam yang `allowedPorts` ditentukan dalam token otentikasi.

Protokol

Lambda MicroVMS mendukung HTTP/2,, gRPC WebSockets, dan SSE melalui URL titik akhir.

Untuk WebSocket koneksi, lewati token otentikasi dan port target melalui subprotokol:

```
// JavaScript WebSocket example
const protocols = [
  "lambda-microvms",                              // Required base protocol
  "lambda-microvms.authentication.<auth-token>",  // Auth token
  "lambda-microvms.port.9000"                     // Target port
];
const ws = new WebSocket('wss://<microvm-endpoint>/path', protocols);
```

Lambda menghapus MicroVM-specific subprotokol dari permintaan sebelum meneruskannya ke aplikasi Anda.

Contoh SDK

Contoh berikut menunjukkan cara menjalankan microVM dan menghubungkannya menggunakan AWS SDK.

Python

Example Contoh - Menjalankan microVM dan menghubungkan dengan boto3

```
import boto3, requests
client = boto3.client("lambda-microvms")
run_resp = client.run_microvm(
    imageIdentifier="arn:aws:lambda:us-east-1:123456789012:microvm-image:my-microvm-
image",
    idlePolicy={"autoResumeEnabled": True, "maxIdleDurationSeconds": 900,
 "suspendedDurationSeconds": 300}
)
microvm_id = run_resp["microvmId"]
endpoint = run_resp["endpoint"]
print(f"MicroVM {microvm_id} running at {endpoint}")
token_resp = client.create_microvm_auth_token(
    microvmIdentifier=microvm_id, expirationInMinutes=30, allowedPorts=[{"allPorts":
 {}}]
)
token = token_resp["authToken"]["X-aws-proxy-auth"]
```

```
resp = requests.get(f"https://{endpoint}/health", headers={"X-aws-proxy-auth":
 token})
print(resp.status_code, resp.json())
```

Node.js

Example Contoh - Menjalankan microVM dan menghubungkan dengan AWS SDK untuk JavaScript

```
import { LambdaMicrovmsClient, RunMicrovmCommand, CreateMicrovmAuthTokenCommand }
 from "@aws-sdk/client-lambda-microvms";
const client = new LambdaMicrovmsClient({});
const { microvmId, endpoint } = await client.send(new RunMicrovmCommand({
  imageIdentifier: "arn:aws:lambda:us-east-1:123456789012:microvm-image:my-microvm-
image",
  idlePolicy: { autoResumeEnabled: true, maxIdleDurationSeconds: 900,
 suspendedDurationSeconds: 300 }
}));
const { authToken } = await client.send(new CreateMicrovmAuthTokenCommand({
  microvmIdentifier: microvmId, expirationInMinutes: 30, allowedPorts: [{ allPorts:
 {} }]
}));
const resp = await fetch(`https://${endpoint}/health`, {
  headers: { "X-aws-proxy-auth": authToken["X-aws-proxy-auth"] }
});
console.log(await resp.json());
```

Mengirim permintaan

Bash

Example Contoh - Mengirim permintaan dengan cURL

```
curl 'https://<microvm-endpoint>' \
  -H 'X-aws-proxy-auth: <TOKEN>' \
  -H 'X-aws-proxy-port: 8080'
```

Python

Example Contoh — Mengirim permintaan dengan perpustakaan permintaan

```
import requests
```

```
response = requests.get('https://<microvm-endpoint>', headers={'X-aws-proxy-auth':
 '<TOKEN>'})
print(response.text)
```

Node.js

Example Contoh — Mengirim permintaan dengan fetch

```
const response = await fetch('https://<microvm-endpoint>', {
  headers: { 'X-aws-proxy-auth': '<TOKEN>', 'X-aws-proxy-port': '8080' }
});
console.log(await response.text());
```

Pengait siklus hidup

Pengait siklus hidup memungkinkan Anda menjalankan logika kustom pada titik-titik kunci dalam siklus hidup microVM — saat dimulai, ditangguhkan, dilanjutkan, atau dihentikan. Gunakan kait untuk menginisialisasi status per penyewa, membersihkan data sebelum ditangguhkan, menyegarkan kredenSIAL pada resume, atau membersihkan sumber daya sebelum penghentian.

Setiap hook adalah titik akhir HTTP yang diekspos aplikasi Anda. Lambda mengirimkan permintaan POST ke hook pada peristiwa siklus hidup yang sesuai. Hooks mendengarkan di jalur `/aws/lambda-microvms/runtime/v1/<hook-name>` pada port yang Anda konfigurasikan.

MicroVM Anda mulai menerima lalu lintas eksternal setelah `/run` hook mengembalikan HTTP 200. Sampai saat itu, titik akhir tidak meneruskan permintaan ke aplikasi Anda.

Kait	Saat dipanggil	Tujuan
`/aws/lambda-microvms/runtime/v1/run`	Setelah MicroVM dimulai dari snapshot	Inisialisasi status per penyewa, atur ulang nilai unik, lakukan pemeriksaan kesehatan. Lalu lintas dimulai setelah hook ini kembali.
`/aws/lambda-microvms/runtime/v1/resume`	Setelah MicroVM dilanjutkan dari status ditangguhkan	Re-establish koneksi jaringan, refresh kredensi, memvalidasi status. MicroVM tetap dalam `SUSPENDED` keadaan saat hook ini dieksekusi; itu beralih ke `RUNNING` setelah hook kembali.

Kait	Saat dipanggil	Tujuan
`/aws/lambda-microvms/runtime/v1/suspend`	Sebelum MicroVM ditangguhkan	Flush penulisan tertunda, tutup koneksi, rilis sumber daya.
`/aws/lambda-microvms/runtime/v1/terminate`	Sebelum MicroVM berakhir	Siram data, beri tahu sistem eksternal, bersihkan. Pengait ini hanya berjalan pada penghentian akhir, bukan saat menangguhkan. Gunakan `/suspend` kait untuk pembersihan pra-penangguhan.

Untuk kait yang berjalan selama pembuatan gambar (`/ready`dan`/validate`), lihatKait pembuatan gambar MicroVM.

Spesifikasi OpenAPI:

```
{
  "openapi": "3.0.2",
  "info": {
    "title": "Lambda MicroVMs Application Hook Interface",
    "version": "2025-12-03"
  },
  "paths": {
    "/ready": {
      "post": {
        "description": "Called by Lambda during MicroVM image creation to determine if
 the application has initialized.",
        "operationId": "Ready",
        "responses": {
          "200": { "description": "Successful invocation." },
          "503": { "description": "Application is not yet ready. Lambda retries until
 timeout." }
        }
      }
    },
    "/resume": {
      "post": {
        "description": "Called by Lambda when resuming a MicroVM that is in the
 SUSPENDED state.",
        "operationId": "Resume",
```

```
        "responses": {
          "200": { "description": "Successful invocation." }
        }
      }
    },
    "/run": {
      "post": {
        "description": "Called by Lambda when a new MicroVM is run from a MicroVM
image.",
        "operationId": "Run",
        "requestBody": {
          "content": {
            "application/json": {
              "schema": { "$ref": "#/components/schemas/RunRequestContent" }
            }
          }
        },
        "responses": {
          "200": { "description": "Successful invocation." }
        }
      }
    },
    "/suspend": {
      "post": {
        "description": "Called by Lambda when suspending a MicroVM.",
        "operationId": "Suspend",
        "responses": {
          "200": { "description": "Successful invocation." }
        }
      }
    },
    "/terminate": {
      "post": {
        "description": "Called by Lambda when terminating a MicroVM, before resources
are released.",
        "operationId": "Terminate",
        "responses": {
          "200": { "description": "Successful invocation." }
        }
      }
    },
    "/validate": {
      "post": {
```

```
        "description": "Called by Lambda when running a MicroVM to validate the
  image build. Use this hook to perform tests that validate your application behaves
  correctly when running. Lambda also samples the portions of the image that are used
  when handling this request, allowing Lambda to prefetch those portions of the image to
  reduce latency at run time.",
        "operationId": "Validate",
        "responses": {
          "200": { "description": "Successful invocation." },
          "503": { "description": "Validation in progress. Lambda retries until
  timeout." }
        }
      }
    }
  },
  "components": {
    "schemas": {
      "RunRequestContent": {
        "type": "object",
        "properties": {
          "microvmId": {
            "type": "string",
            "description": "The MicroVM identifier."
          },
          "runHookPayload": {
            "type": "string",
            "description": "Run hook payload provided to RunMicrovm."
          }
        }
      }
    }
  },
  "servers": [
    { "url": "/aws/lambda-microvms/runtime/v1" }
  ]
}
```

Menangguhkan dan Melanjutkan MicroVM

Tangguhkan MicroVM untuk mengurangi biaya sambil mempertahankan status aplikasi. Saat berlari, Anda membayar biaya komputasi. Saat ditangguhkan, Anda hanya membayar biaya penyimpanan snapshot.

Cara menangguhkan

Ada dua cara untuk menangguhkan microVM:

1. Kebijakan idle (otomatis) — Konfigur `maxIdleDurationSeconds` asikan dalam kebijakan idle. Jika tidak ada lalu lintas yang tiba di titik akhir microVM untuk durasi tersebut, Lambda menangguhkan microVM secara otomatis.
2. Panggilan API (eksplisit) — Panggilan `suspend-microvm` untuk segera menangguhkan:

```
aws lambda-microvms suspend-microvm --microvm-identifier microvm-id
```

Pengait /suspend

Sebelum menangguhkan, Lambda memanggil hook Anda`/suspend`. Gunakan untuk menghapus penulisan yang tertunda, menutup koneksi jaringan, dan melepaskan sumber daya yang tidak boleh bertahan melintasi batas penangguhan.

Lanjutkan perilaku

Ketika microVM dilanjutkan (melalui panggilan API atau auto-resume), Lambda mengembalikan memori dan status disk dari pos pemeriksaan suspensi. MicroVM tetap dalam `SUSPENDED` keadaan saat `/resume` hook dieksekusi. Setelah hook mengembalikan HTTP 200, microVM bertransisi ke `RUNNING` dan mulai menerima lalu lintas.

Gunakan `/resume` hook untuk menyegarkan kredenSIAL, membangun kembali koneksi jaringan, dan memvalidasi status.

```
aws lambda-microvms resume-microvm --microvm-identifier microvm-id
```

Auto-resume

Ketika `autoResumeEnabled=true` lalu lintas tiba di titik akhir microVM yang ditangguhkan, Lambda melanjutkan microVM secara otomatis. Lambda menahan permintaan masuk saat resume selesai (termasuk `/resume` hook), lalu mengirimkannya ke aplikasi Anda.

Resume menambahkan latensi ke permintaan pertama. Durasi tergantung pada ukuran keadaan tersuspensi yang dipulihkan dan durasi `/resume` kait Anda.

Jika resume tidak berhasil, Lambda mengembalikan 502 Bad Gateway ke penelepon.

> **Note**
>
> Auto-resume menambahkan latensi hanya ke permintaan pertama setelah penangguhan. Permintaan selanjutnya saat microVM berjalan tidak terpengaruh.

Penskalaan dan konkurensi

Anda membuat microVM baru dengan menelepon. `run-microvm` Setiap microVM memiliki titik akhir khusus sendiri. Tidak ada penyeimbangan beban di seluruh microVM dari satu titik akhir.

Account-level kapasitas — Akun Anda memiliki kuota untuk total memori yang dapat dialokasikan di semua microVM Anda di `SUSPENDED` negara bagian `RUNNING` atau di Wilayah, dan Anda dapat menskalakan secara vertikal hingga empat kali kuota ini. Untuk meminta peningkatan kuota, kunjungi konsol Kuota Layanan dan cari MicroVMS Lambda.

Model biaya:

- Menjalankan MicroVMS dikenakan biaya komputasi.
- MicroVM yang ditangguhkan dikenakan biaya penyimpanan snapshot tetapi tidak dikenakan biaya komputasi.
- MicroVM yang dihentikan tidak dikenakan biaya.

Strategi untuk mengelola kapasitas:

- Menangguhkan microVM idle — Konfigurasikan kebijakan idle untuk secara otomatis menangguhkan microVM yang tidak menerima lalu lintas.
- Hentikan MicroVM yang tidak lagi diperlukan — Gunakan `suspendedDurationSeconds` untuk menghentikan secara otomatis setelah durasi maksimum yang ditangguhkan, atau menelepon secara eksplisit. `terminate-microvm`
- Right-size kebijakan idle — Tetapkan `maxIdleDurationSeconds` berdasarkan pola lalu lintas Anda. Waktu idle yang lebih pendek membebaskan kapasitas lebih cepat.

Mengakhiri MicroVM

Hentikan microVM ketika tidak lagi diperlukan. Pengakhiran melepaskan semua sumber daya komputasi dan menghentikan semua biaya.

Sebelum melepaskan sumber daya, Lambda memanggil `/terminate` hook Anda. Gunakan untuk membersihkan data yang tertunda atau memberi tahu sistem eksternal. Peng `/terminate` ait hanya berjalan pada penghentian akhir - tidak berjalan ketika microVM ditangguhkan.

```
aws lambda-microvms terminate-microvm --microvm-identifier microvm-id
```

Daftar MicroVM

Daftar semua MicroVM di akun Anda, secara opsional difilter berdasarkan gambar:

```
aws lambda-microvms list-microvms

# Filter by image
aws lambda-microvms list-microvms --image-identifier my-image --image-version 1.0
```

Penanganan kesalahan

Jalankan kesalahan

Tabel berikut mencantumkan kesalahan umum yang dikembalikan oleh `run-microvm` API:

Kesalahan	Penyebab	Solusi
`ServiceQuotaExceededException`	Akun telah mencapai kuota memori untuk microVM bersamaan.	Hentikan microVM yang tidak aktif atau minta peningkatan kuota.
`ResourceNotFoundException`	Gambar yang ditentukan tidak ada atau tidak dalam `CREATED` keadaan.	Verifikasi pengenal gambar dan konfirmasikan pembuatan selesai.
`ValidationException`	Satu atau lebih parameter permintaan tidak valid.	Periksa nilai kebijakan idle, format pengidentifikasi gambar, dan ARN konektor.
`ThrottlingException`	Batas tarif API untuk operasi ini telah terlampaui.	Terapkan backoff eksponensial dengan jitter.

Strategi percobaan kembali

Untuk kesalahan transien (`ThrottlingException`,`InternalServerException`), gunakan backoff eksponensial:

```
import time, random
def run_with_retry(client, params, max_retries=5):
    for attempt in range(max_retries):
        try:
            return client.run_microvm(**params)
        except client.exceptions.ThrottlingException:
            delay = (2 ** attempt) + random.uniform(0, 1)
            time.sleep(delay)
    raise Exception("Max retries exceeded")
```

Untuk informasi tentang integrasi layanan yang didukung dengan Instans Terkelola Lambda, lihat. the section called “Integrasi”

Jaringan

Anda mengonfigurasi akses jaringan untuk AWS Lambda microVMS Anda dengan mengaitkan sumber daya Konektor Jaringan dengan microVM Anda pada waktu berjalan. Konektor jaringan ditentukan saat Anda menelepon `run-microvm` dan tidak dapat diubah saat microVM sedang berjalan.

Gambaran umum

Setiap microVM dapat memiliki konfigurasi jaringan ingress (inbound) dan egress (outbound) independen:

- Konektor jaringan masuk memungkinkan konektivitas masuk. Klien terhubung ke titik akhir HTTPS yang dikelola layanan, dan Lambda meneruskan lalu lintas ke port yang Anda konfigurasikan dalam microVM. Konektor Ingress AWS dikelola - Anda mereferensikannya dengan ARN saat menjalankan microVM.
- Konektor jaringan keluar memungkinkan lalu lintas keluar. Secara default, microVMS memiliki akses internet publik. Anda dapat membuat konektor keluar VPC yang dikelola pelanggan untuk merutekan lalu lintas keluar melalui VPC Anda sebagai gantinya.

Konektor tunggal dapat digunakan kembali di banyak microVM - ini adalah pola penggunaan yang dimaksudkan.

Konektivitas Masuk

Setiap MicroVM Lambda dapat dijangkau di URL titik akhir HTTPS unik, yang ditetapkan saat Anda menelepon. `run-microvm` Klien mengirim permintaan ke titik akhir ini melalui HTTPS. Lambda merutekan setiap permintaan ke port di dalam microVM Anda, tempat aplikasi Anda menerimanya.

Secara default, permintaan yang diterima di titik akhir dirutekan ke port 8080 di dalam microVM. Untuk rute ke port yang berbeda, lihatthe section called “Perutean port”.

Protokol berikut didukung pada titik akhir masuk:

- HTTP/1.1
- HTTP/2
- WebSockets
- gRPC
- Server-Sent Acara (SSE)

> **Note**
>
> Lalu lintas antara klien Anda dan endpoint microVM selalu dienkripsi dengan TLS. Aplikasi Anda dapat melayani permintaan melalui HTTP atau HTTPS secara internal.

Perutean port

Lambda memilih port target di dalam microVM Anda menggunakan urutan prioritas berikut:

1. **`X-aws-proxy-port`**header — Untuk permintaan HTTP standar, sertakan header ini dengan nomor port target.
2. WebSocket subprotokol - Jika WebSocket klien Anda tidak dapat mengatur header khusus, tentukan port target sebagai subprotokol bernama`lambda-microvms.port.`*N*, di mana nomor *N* port. Anda menyediakan subprotokol saat Anda membuka koneksi. WebSocket Sebagai contoh, lihat the section called “Protokol”.

3. Default (8080) - Jika tidak ada yang ditentukan, permintaan rute ke port 8080.

> **Important**
>
> Port target harus berada dalam yang `allowedPorts` ditentukan dalam token otentikasi. Permintaan ke port yang tidak sah menerima respons 403 Terlarang.

Autentikasi

Semua permintaan ke endpoint microVM memerlukan token otentikasi yang valid di header. `X-aws-proxy-auth` Anda menghasilkan token menggunakan`create-microvm-auth-token`. Setiap token adalah string JWE (JSON Web Encryption) terenkripsi yang dicakup ke:

- MicroVM tertentu (diidentifikasi oleh ID).
- Satu set port yang diizinkan (port tunggal, jangkauan, atau semua port).
- Waktu kedaluwarsa (dikonfigurasi pada pembuatan token).

Contoh berikut membuat token dan menggunakannya untuk mengirim permintaan yang diautentikasi:

```
aws lambda-microvms create-microvm-auth-token \
  --microvm-identifier microvm-id \
  --expiration-in-minutes 30 \
  --allowed-ports '[{"port":8080}]'
```

```
curl 'https://microvm-endpoint' \
  -H 'X-aws-proxy-auth: TOKEN' \
  -H 'X-aws-proxy-port: 8080'
```

Untuk panduan lengkap dalam membuat token dan menghubungkan ke microVM, termasuk WebSocket koneksi, lihat. the section called “Menghubungkan ke MicroVM”

Respons kesalahan

Kode status HTTP berikut dikembalikan oleh endpoint microVM ketika tidak dapat memproses atau mengirimkan permintaan ke aplikasi Anda. Tanggapan ini berasal dari titik akhir, bukan dari aplikasi Anda.

Kode	Status	Penyebab dan resolusi
400	Permintaan Buruk	Permintaan cacat, atau header port atau subprotokol yang tidak valid. WebSocket Verifikasi formatnya.
403	Dilarang	Token yang hilang, kedaluwarsa, atau tidak valid; atau port yang diminta tidak ada dalam token. `allowedPorts` Buat token baru, atau gunakan port yang diizinkan.
429	Terlalu Banyak Permintaan	Batas nilai terlampaui (tingkat akun atau per-microVM). Coba lagi dengan backoff eksponensial.
500	Kesalahan Server Internal	Terjadi kesalahan internal. Coba lagi permintaannya.
502	Gateway Buruk	Aplikasi tidak merespons, atau resume otomatis tidak berhasil dalam jumlah maksimum upaya coba lagi. Lihat the section called “Auto-resume”.

Minta header

Namespace `X-aws-proxy-*` header dicadangkan oleh Lambda untuk metadata permintaan, seperti token otentikasi () dan port target ()`X-aws-proxy-auth`. `X-aws-proxy-port` Lambda menghapus `X-aws-proxy-*` header sebelum meneruskan permintaan ke aplikasi Anda.

Request/response bandwidth

Setiap microVM Lambda memiliki request/response bandwidth yang berskala linier dengan ukurannya. Bandwidth ini berlaku untuk semua lalu lintas melalui endpoint microVM, baik permintaan masuk maupun respons keluar.

Ukuran microVM (baseline)	Bandwidth maks
0,5 GB, 0,25 vCPU	1 MB/s (8 Mbps)
1 GB, 0,5 vCPU	2 MB/s (16 Mbps)
2 GB, 1 vCPU	4 MB/s (32 Mbps)

Ukuran microVM (baseline)	Bandwidth maks
4 GB, 2 vCPU	8 MB/s (64 Mbps)
8 GB, 4 vCPU	16 MB/s (128 Mbps)

Jika Anda mengalami peningkatan latensi permintaan karena saturasi jaringan, kurangi konkurensi permintaan atau ukuran muatan, atau pilih ukuran microVM yang lebih besar untuk meningkatkan bandwidth yang tersedia.

HTTP/2 dukungan

Lambda MicroVMS mendukung HTTP/2 pada titik akhir masuk. Lambda menegosiasikan protokol melalui ALPN (Application-Layer Protocol Negotiation) selama jabat tangan TLS, lebih memilih dan kembali ke. HTTP/2 HTTP/1.1 HTTP/2-capable Klien menggunakannya secara otomatis.

Untuk menggunakan HTTP/2 antara titik akhir dan aplikasi Anda di dalam microVM:

- Aplikasi Anda melayani TLS - Lambda HTTP/2 bernegosiasi dengan aplikasi Anda melalui ALPN, kembali HTTP/1.1 ke HTTP/2 jika tidak didukung.
- Aplikasi Anda menyajikan HTTP teks biasa — Sertakan `X-aws-proxy-force-h2: true` header dalam permintaan Anda untuk digunakan HTTP/2 pada koneksi ke aplikasi Anda.

Konektivitas keluar

Secara default, Lambda MicroVMS memiliki akses internet publik di jalur keluar. Untuk menghubungkan microVM dengan sumber daya di VPC pribadi Anda — seperti RDS, API internal ElastiCache, dan sistem lokal melalui Direct Connect atau VPN — buat Konektor Jaringan Lambda dengan konfigurasi VPC Anda.

Saat menggunakan jalan keluar VPC, lalu lintas keluar tunduk pada aturan grup keamanan dan ACL jaringan yang mengatur lalu lintas di VPC Anda.

Bekerja dengan konektor jaringan jalan keluar

Konektor jaringan keluar merutekan lalu lintas keluar dari microVM Anda melalui VPC Anda. Anda membuat konektor sekali, lalu mereferensikannya dengan ARN saat memulai microVMS melalui perintah. `run-microvm`

Prasyarat

Sebelum membuat konektor jaringan, Anda memerlukan peran IAM yang memungkinkan Lambda membuat antarmuka jaringan elastis (ENI) di VPC Anda. Peran tersebut memerlukan izin berikut:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "CreateENI",
      "Effect": "Allow",
      "Action": "ec2:CreateNetworkInterface",
      "Resource": [
        "arn:aws:ec2:*:*:network-interface/*",
        "arn:aws:ec2:*:*:subnet/*",
        "arn:aws:ec2:*:*:security-group/*"
      ]
    },
    {
      "Sid": "TagENI",
      "Effect": "Allow",
      "Action": "ec2:CreateTags",
      "Resource": "arn:aws:ec2:*:*:network-interface/*",
      "Condition": {
        "StringEquals": {
          "ec2:ManagedResourceOperator": "network-connectors.lambda.amazonaws.com"
        }
      }
    }
  ]
}
```

Membuat konektor jaringan

Buat konektor dengan menentukan subnet VPC Anda, grup keamanan, dan protokol `IPv4` jaringan (atau): `DualStack`

```
aws lambda-core create-network-connector \
  --name my-connector \
  --configuration '{
    "VpcEgressConfiguration": {
      "SubnetIds": ["subnet-xxx"],
```

```
      "SecurityGroupIds": ["sg-xxx"],
      "NetworkProtocol": "IPv4",
      "AssociatedComputeResourceTypes": ["MicroVm"]
    }
  }' \
  --operator-role arn:aws:iam::123456789012:role/NetworkConnectorOperatorRole
```

Status konektor jaringan

Konektor harus dalam `ACTIVE` keadaan sebelum Anda dapat mereferensikannya`run-microvm`.

Status	Deskripsi
`PENDING`	Konektor sedang dibuat (ENI yang mendasari sedang disediakan).
`ACTIVE`	Konektor siap digunakan.
`INACTIVE`	Konektor sementara tidak aktif.
`FAILED`	Penyediaan atau pembaruan gagal. Periksa `StateReason` .
`DELETING`	Konektor sedang dihapus; ENI sedang dibersihkan.
`DELETE_FAILED`	Penghapusan gagal.

Versi konektor jaringan

Pembuatan versi berlaku untuk konektor jaringan keluar VPC (tipe konektor). `VPC_EGRESS` Konektor keluar VPC diberi versi: setiap `update-network-connector` panggilan yang berhasil membuat versi baru, dan versi sebelumnya tetap tersedia.

Saat Anda menjalankan microVM, Anda mereferensikan konektor keluar VPC oleh ARN di. `--egress-network-connectors` Anda dapat menggunakan salah satu bentuk:

- ARN yang tidak memenuhi syarat - menyelesaikan ke versi konektor terbaru. MicroVM mengambil perubahan konfigurasi secara otomatis. `arn:aws:lambda:us-east-1:123456789012:network-connector:nc-id`
- ARN yang memenuhi syarat — menyematkan microVM ke versi tertentu. Gunakan ini untuk menahan microVM pada konfigurasi yang diketahui untuk pengujian, pemecahan masalah,

atau rollback. `arn:aws:lambda:us-east-1:123456789012:network-connector:nc-id:version-number`

MicroVM pindah ke versi baru hanya ketika Anda mengubah referensi ARN, sehingga Anda dapat mengadopsi versi baru sesuai jadwal Anda sendiri.

Contoh `run-microvm` panggilan menyematkan versi konektor tertentu:

```
aws lambda-microvms run-microvm \
  --image-identifier arn:aws:lambda:us-east-1:123456789012:microvm-image:my-microvm-image \
  --egress-network-connectors arn:aws:lambda:us-east-1:123456789012:network-connector:nc-id:version-number \
  --idle-policy
 '{"maxIdleDurationSeconds":900,"suspendedDurationSeconds":1800,"autoResumeEnabled":false}'
```

Menjalankan microVM dengan konektor jaringan

Referensi konektor ARN saat menjalankan microVM:

```
aws lambda-microvms run-microvm \
  --image-identifier arn:aws:lambda:us-east-1:123456789012:microvm-image:my-microvm-image \
  --egress-network-connectors connector-arn \
  --idle-policy
 '{"maxIdleDurationSeconds":900,"suspendedDurationSeconds":1800,"autoResumeEnabled":false}'
```

> **Note**
>
> Sebelum Anda memperbarui atau menghapus konektor, pastikan semua microVMS yang menggunakannya telah dihentikan. Memodifikasi konektor yang aktif digunakan dapat menyebabkan masalah konektivitas jaringan untuk menjalankan microVMS.

Menggunakan MicroVMS Lambda dengan titik akhir VPC antarmuka (AWS PrivateLink)

Anda dapat menggunakan AWS PrivateLink konektivitas pribadi melalui AWS jaringan antara sumber daya VPC Anda dan microVM Lambda, tanpa melintasi internet publik. MicroVMS mendukung dua titik akhir VPC, tergantung pada tujuan lalu lintas yang diinginkan:

- API manajemen microVM (membuat gambar, menjalankan, menangguhkan, menghentikan) — menggunakan titik akhir VPC Lambda () yang ada. `com.amazonaws.region.lambda`
- Konektivitas ke microVMS (lalu lintas HTTPS ke aplikasi yang sedang berjalan) — menggunakan endpoint () terpisah. `com.amazonaws.region.lambda-microvm`

Titik akhir VPC untuk API manajemen microVM

MicroVMS Lambda berbagi layanan titik akhir VPC yang sama dengan Lambda (). `com.amazonaws.region.lambda` Untuk petunjuk selengkapnya, lihat Membuat titik akhir antarmuka untuk Lambda.

Kebijakan titik akhir untuk API manajemen microVM

Untuk mengontrol siapa yang dapat menggunakan titik akhir antarmuka Anda dan tindakan API MicroVMS Lambda mana yang dapat mereka lakukan, lampirkan kebijakan titik akhir. Kebijakan menentukan prinsip yang dapat melakukan tindakan, tindakan yang dapat mereka lakukan, dan sumber daya yang dapat mereka lakukan. Tindakan microVMS Lambda menggunakan awalan tindakan IAM. `lambda:`

Untuk informasi selengkapnya, lihat Mengontrol Akses ke Layanan dengan titik akhir VPC dalam Panduan Pengguna Amazon VPC.

Contoh kebijakan berikut memungkinkan pengguna `MyUser` untuk membuat daftar dan mendapatkan gambar microVM melalui titik akhir:

```
{
  "Statement": [
    {
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:user/MyUser"
      },
```

```
      "Effect": "Allow",
      "Action": [
        "lambda:ListMicrovmImages",
        "lambda:GetMicrovmImage"
      ],
      "Resource": "*"
    }
  ]
}
```

Titik akhir VPC untuk konektivitas microVM

Untuk menjaga lalu lintas HTTPS ke microVMS yang sedang berjalan Anda tetap pribadi, buat titik akhir antarmuka untuk layanan. `com.amazonaws.region.lambda-microvm` Titik akhir ini menangani koneksi ke URL endpoint microVM (misalnya,). `abc123def456.lambda-microvm.us-east-1.on.aws`

Untuk mempelajari lebih lanjut tentang properti titik akhir antarmuka, tinjau panduan titik akhir antarmuka dalam dokumentasi VPC Amazon.

Membuat titik akhir

Untuk membuat titik akhir antarmuka untuk konektivitas microVM (konsol)

1. Buka Halaman titik akhir konsol Amazon VPC.
2. Pilih Buat titik akhir.
3. Untuk Kategori layanan, verifikasi bahwa AWS layanan dipilih.
4. Untuk Nama Layanan, pilih`com.amazonaws.region.lambda-microvm`. Verifikasi bahwa Type adalah Interface.
5. Pilih VPC dan subnet.
6. Untuk mengaktifkan DNS pribadi untuk titik akhir antarmuka, pilih kotak centang Aktifkan nama DNS (disarankan). Ini memastikan bahwa permintaan yang menggunakan nama host endpoint microVM publik secara otomatis diselesaikan ke titik akhir antarmuka Anda, tanpa perlu perubahan sisi klien.
7. Untuk Grup keamanan, pilih satu grup keamanan atau lebih. Grup keamanan harus mengizinkan lalu lintas TCP keluar pada port 443 ke antarmuka jaringan titik akhir.
8. Pilih Buat titik akhir.

Untuk menggunakan opsi DNS pribadi, Anda harus mengatur `enableDnsHostnames` dan `enableDnsSupport` atribut VPC Anda. Untuk informasi selengkapnya, lihat Melihat dan memperbarui dukungan DNS untuk VPC Anda di Panduan Pengguna Amazon VPC.

Untuk membuat titik akhir antarmuka untuk konektivitas microVM (CLI)AWS

```
aws ec2 create-vpc-endpoint \
  --vpc-id vpc-ec43eb89 \
  --vpc-endpoint-type Interface \
  --service-name com.amazonaws.us-east-1.lambda-microvm \
  --subnet-id subnet-abababab \
  --security-group-id sg-1a2b3c4d \
  --private-dns-enabled
```

Untuk memverifikasi bahwa titik akhir tersedia dan DNS pribadi berlaku:

```
aws ec2 describe-vpc-endpoints \
  --vpc-endpoint-ids vpce-1a2b3c4d5e6f7g8h9 \
  --query 'VpcEndpoints[0].
{State:State,PrivateDns:PrivateDnsEnabled,Dns:DnsEntries[*].DnsName}'
```

Perilaku DNS pribadi

Saat DNS pribadi diaktifkan — Titik akhir mengelola resolusi DNS untuk di dalam VPC `*.lambda-microvm.region.on.aws` Anda. Nama host endpoint microVM Anda yang ada (misalnya,`abc123def456.lambda-microvm.us-east-1.on.aws`) diselesaikan ke alamat IP pribadi dari antarmuka jaringan titik akhir. Tidak diperlukan perubahan klien.

Saat DNS pribadi dinonaktifkan - Amazon VPC menghasilkan nama DNS khusus titik akhir untuk titik akhir Anda dalam formulir. `vpce-id-hash.lambda-microvm.region.vpce.amazonaws.com` Untuk merutekan lalu lintas melalui titik akhir ini sambil tetap mencapai microVM yang benar, Anda harus mempertahankan nama host microVM asli di dua tempat:

- TLS Server Name Indication (SNI) — Jabat tangan TLS menggunakan nilai ini untuk mengidentifikasi microVM mana koneksi itu.
- Header Host HTTP — Proxy menggunakan nilai ini untuk merutekan permintaan ke microVM yang benar.

Jika salah satu nilai disetel ke nama host titik akhir VPC alih-alih nama host microVM, koneksi tidak dapat dirutekan ke microVM yang benar.

Contoh: Connect melalui nama DNS khusus titik akhir

Contoh berikut menggunakan curl dengan `--connect-to` flag untuk mengarahkan koneksi TCP ke titik akhir VPC Anda sambil menyimpan nama host microVM di URL, SNI, dan header Host:

```
ENDPOINT_HOST=abc123def456.lambda-microvm.us-east-1.on.aws
VPCE_HOST=vpce-0a1b2c3d4e5f67890-a1b2c3d4.lambda-microvm.us-east-1.vpce.amazonaws.com

curl --connect-to "$ENDPOINT_HOST:443:$VPCE_HOST:443" \
  -H "x-aws-proxy-auth: $MICROVM_AUTH_TOKEN" \
  -H "x-aws-proxy-port: 8080" \
  "https://$ENDPOINT_HOST/"
```

`--connect-to`Bendera memberi tahu curl untuk membuka koneksi TCP ke alamat titik akhir VPC, sedangkan URL, TLS SNI, dan header Host tetap disetel ke nama host microVM Anda.

Untuk informasi selengkapnya, lihat Mengakses layanan melalui titik akhir antarmuka dalam Panduan Pengguna Amazon VPC.

Kebijakan titik akhir untuk konektivitas microVM

Anda dapat melampirkan kebijakan titik akhir untuk mengontrol microVM mana yang dapat dijangkau melalui titik akhir VPC. `lambda-microvm` Kebijakan endpoint pada `lambda-microvm` layanan memungkinkan Anda menjangkau koneksi ke akun atau organisasi tertentu. Secara default, titik akhir memungkinkan koneksi ke microVMS di akun apa pun. AWS (Catatan: Klien yang membuat koneksi harus tetap memegang token autentikasi microVM yang valid untuk diberikan akses).

Secara default, titik akhir VPC memiliki kebijakan akses penuh yang memungkinkan semua lalu lintas. Saat Anda mengganti kebijakan default dengan kebijakan khusus, Lambda MicroVMS mengevaluasi kebijakan tersebut terhadap `lambda:ConnectMicrovm` tindakan pada setiap koneksi yang dilakukan melalui titik akhir. Jika kebijakan tidak mengizinkan penyambungan ke microVMS, sambungan akan ditolak dengan respons HTTP 403 Forbidden. Kebijakan yang tidak memberikan `lambda:ConnectMicrovm` menolak semua koneksi melalui titik akhir.

> **Note**
>
> `lambda:ConnectMicrovm`Tindakan ini mengotorisasi koneksi ke titik akhir microVM melalui titik akhir antarmuka. Ini bukan operasi API Lambda dan tidak dapat digunakan dalam kebijakan berbasis identitas IAM atau berbasis sumber daya — ini hanya valid dalam kebijakan titik akhir VPC.

Prinsipal dan sumber daya

Koneksi ke endpoint microVM diautentikasi dengan token autentikasi MicroVM daripada Signature Version 4. AWS Untuk alasan ini, tidak ada prinsipal IAM yang terkait dengan koneksi. Sebaliknya, Lambda MicroVMS mengevaluasi kebijakan endpoint dengan prinsipal anonim. Ini berarti:

- `Principal`harus`"*"`. Kebijakan yang menyebutkan kepala sekolah tertentu tidak cocok dan menyangkal setiap koneksi.
- Kunci kondisi yang bergantung pada identitas pemohon (seperti`aws:PrincipalArn`,`aws:PrincipalOrgID`, dan`aws:userid`) tidak diisi dan tidak akan cocok.
- `Resource`juga harus`"*"`. MicroVM Lambda tidak mencakup evaluasi kebijakan titik akhir untuk ARN sumber daya microVM individu. Untuk membatasi microVM mana yang dapat dicapai titik akhir, gunakan kunci `aws:ResourceAccount` kondisi daripada elemen. `Resource`

Kunci kondisi yang didukung

Kunci syarat	Deskripsi
`aws:ResourceAccount`	AWS Akun yang memiliki microVM yang terhubung ke.
`aws:ResourceOrgID`	ID AWS organisasi Organizations dari akun yang memiliki microVM.
`aws:SourceVpce`	ID titik akhir antarmuka yang dilewati koneksi.
`aws:SourceVpc`	ID dari VPC koneksi berasal dari.
`aws:VpcSourceIp`	Alamat IP pribadi klien yang membuat koneksi.

Contoh: Izinkan koneksi hanya ke microVMS di akun Anda sendiri

Kebijakan endpoint berikut memungkinkan koneksi melalui titik akhir hanya ke microVM yang dimiliki oleh akun 111122223333. Koneksi ke microVMS yang dimiliki oleh akun lain ditolak.

```
{
  "Statement": [
    {
```

```
      "Principal": "*",
      "Effect": "Allow",
      "Action": "lambda:ConnectMicrovm",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceAccount": "111122223333"
        }
      }
    }
  ]
}
```

Keamanan dan izin

Bagian ini menjelaskan peran IAM, token otentikasi, dan kontrol akses untuk AWS Lambda microVM.

Peran IAM

Membangun peran

Peran ini digunakan selama pembuatan gambar. Ini membutuhkan izin berikut untuk mengambil artefak sumber dan menghasilkan log:`s3:GetObject`,`logs:CreateLogGroup`,`logs:CreateLogStream`, dan`logs:PutLogEvents`. Tambahkan juga `ecr:GetAuthorizationToken` jika mereferensikan gambar ECR pribadi.

Peran build adalah opsional. Jika Anda tidak memberikan peran build, Lambda tidak dapat menulis log build ke CloudWatch.

Peran eksekusi

Peran ini digunakan pada runtime MicroVM untuk memberikan izin akses untuk menulis log ke CloudWatch dan menggunakan AWS layanan lain.

Peran eksekusi adalah opsional. Jika Anda tidak memberikan peran eksekusi, Lambda tidak memancarkan log runtime ke CloudWatch dan microVM Anda tidak dapat mengakses layanan lain AWS .

Kait siklus hidup dijalankan di bawah peran yang terkait dengan fase mereka. Build-time hook (`/readydan/validate`) dijalankan di bawah peran build. Runtime hook (`/run`,`/resume`,`/suspend`, dan`/terminate`) dijalankan di bawah peran eksekusi.

Kebijakan kepercayaan

Kedua peran memerlukan kebijakan kepercayaan yang memungkinkan prinsipal layanan Lambda `lambda.amazonaws.com` untuk melakukan `sts:AssumeRole` dan`sts:TagSession`.

Token otentikasi

Token mengontrol akses masuk untuk menjalankan microVM. Mereka memiliki cakupan port; setiap token menentukan port mana yang diberikan akses.

```
# Token granting access to all ports
aws lambda-microvms create-microvm-auth-token \
  --microvm-identifier <id> \
  --expiration-in-minutes 30 \
  --allowed-ports '[{"allPorts":{}}]'

# Token granting access to port 8080 only
aws lambda-microvms create-microvm-auth-token \
  --microvm-identifier <id> \
  --expiration-in-minutes 30 \
  --allowed-ports '[{"port":8080}]'
```

Token akses shell

Akses Shell menggunakan API token terpisah. MicroVM target harus dijalankan dengan konektor `SHELL_INGRESS` jaringan (`arn:aws:lambda:us-east-1:aws:network-connector:aws-network-connector:SHELL_INGRESS`). Jika microVM tidak diluncurkan dengan konektor ini, `create-microvm-shell-auth-token` gagal dengan`ValidationException`:

```
aws lambda-microvms create-microvm-shell-auth-token \
  --microvm-identifier <id> \
  --expiration-in-minutes 30
```

Untuk panduan lengkap menghubungkan ke shell microVM, lihatthe section called “Akses shell”.

Menandai sumber daya

Gambar Lambda MicroVM mendukung tag AWS sumber daya untuk organisasi dan alokasi biaya. Tambahkan tag saat membuat gambar menggunakan `--tags` parameter atau secara eksplisit setelah pembuatan. Anda juga dapat membuat daftar dan menghapus tag, seperti yang ditunjukkan di bawah ini: Anda juga dapat mereferensikan tag dalam kondisi kebijakan IAM untuk mengontrol akses ke sumber daya microVM.

```
# Add tags when creating an image
aws lambda-microvms create-microvm-image \
  --name my-app-image \
  --code-artifact '{"uri":"s3://my-bucket/app.zip"}' \
  --base-image-arn arn:aws:lambda:us-east-1:aws:microvm-image:al2023-1 \
  --build-role-arn arn:aws:iam::123456789012:role/MicrovmBuildRole \
  --tags '{"Environment":"production","Team":"platform"}'

# List tags on a resource
aws lambda-microvms list-tags \
  --resource arn:aws:lambda:us-east-1:123456789012:microvm-image:my-app-image

# Add tags to an existing resource
aws lambda-microvms tag-resource \
  --resource arn:aws:lambda:us-east-1:123456789012:microvm-image:my-app-image \
  --tags '{"CostCenter":"12345","Project":"sandbox-platform"}'

# Remove tags by key
aws lambda-microvms untag-resource \
  --resource arn:aws:lambda:us-east-1:123456789012:microvm-image:my-app-image \
  --tag-keys '["CostCenter","Project"]'
```

Referensi izin IAM

Lambda MicroVMS mendefinisikan tindakan IAM berikut. Gunakan ini dalam kebijakan IAM untuk mengontrol akses ke operasi microVM.

Tindakan	Deskripsi
`lambda:CreateMicrovmImage`	Buat gambar microVM baru.
`lambda:UpdateMicrovmImage`	Perbarui gambar microVM yang ada.

Tindakan	Deskripsi
`lambda:DeleteMicrovmImage`	Hapus gambar microVM.
`lambda:GetMicrovmImage`	Dapatkan detail gambar MicroVM.
`lambda:ListMicrovmImages`	Daftar gambar MicroVM di akun.
`lambda:RunMicrovm`	Jalankan microVM baru.
`lambda:GetMicrovm`	Dapatkan status dan detail MicroVM.
`lambda:ListMicrovms`	Daftar MicroVM di akun.
`lambda:SuspendMicrovm`	Menangguhkan microVM yang sedang berjalan.
`lambda:ResumeMicrovm`	Lanjutkan microVM yang ditangguhkan.
`lambda:TerminateMicrovm`	Hentikan microVM.
`lambda:CreateMicrovmAuthToken`	Hasilkan token otentikasi untuk microVM.
`lambda:CreateMicrovmShellAuthToken`	Hasilkan token akses shell.

Format ARN sumber daya

```
# MicroVM image
arn:aws:lambda:<region>:<account>:microvm-image:<image-name>
# Network connector
arn:aws:lambda:<region>:<account>:network-connector:<connector-id>
```

Contoh: Least-privilege kebijakan untuk operator

```
{
    "Version": "2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
```

```
        "Action": [
          "lambda:ListMicrovms",
          "lambda:ListMicrovmImages"
        ],
        "Resource": "*"
      },
      {
        "Effect": "Allow",
        "Action": [
          "lambda:RunMicrovm",
          "lambda:GetMicrovm",
          "lambda:SuspendMicrovm",
          "lambda:ResumeMicrovm",
          "lambda:TerminateMicrovm",
          "lambda:CreateMicrovmAuthToken",
          "lambda:GetMicrovmImage"
        ],
        "Resource": "arn:aws:lambda:*:123456789012:microvm-image:*"
      }
    ]
}
```

Memantau

Halaman ini mencakup CloudWatch logging (build dan runtime) dan CloudTrail audit untuk AWS Lambda microVM.

Pencatatan (CloudWatch Log)

AWS Lambda MicroVMS mengalirkan log ke CloudWatch dalam dua fase:

Membangun log

Lambda menghasilkan log build selama pembuatan gambar, termasuk output dari `Dockerfile` eksekusi, startup aplikasi, dan pemanggilan kait. Log ini dialirkan menggunakan peran build dan ditulis ke grup `/aws/lambda-microvms/<image-name>` log default.

Log runtime

Lambda mengalirkan aplikasi stdout dan stderr dari menjalankan MicroVMS ke. CloudWatch Log runtime dialirkan menggunakan peran eksekusi dan ditulis ke grup log default yang sama, dengan aliran log default ke ID microVM.

> **Important**
>
> Tanpa peran eksekusi, aplikasi stdout dan stderr tidak diteruskan ke saat runtime. CloudWatch Untuk menerima log runtime, berikan peran eksekusi dengan`logs:CreateLogGroup`,`logs:CreateLogStream`, dan `logs:PutLogEvents` izin.

Tujuan log kustom

Anda dapat mengonfigurasi grup log kustom dan streaming dengan menggunakan `--logging` parameter pada `run-microvm` atau`create-microvm-image`:

```
--logging '{"cloudWatch":{"logGroup":"/my-app/microvms","logStream":"custom-stream"}}'
```

Jika Anda menghilangkan`logStream`, default ke ID microVM.

Anda dapat menonaktifkan logging secara eksplisit dengan meneruskan konfigurasi berikut:

```
--logging '{"disabled":{}}'
```

CloudTrail penebangan

Lambda MicroVM terintegrasi dengan CloudTrail, layanan yang menyediakan catatan tindakan yang diambil oleh pengguna, peran, atau layanan. AWS CloudTrail menangkap semua panggilan API untuk Lambda MicroVMS sebagai peristiwa. Panggilan yang ditangkap termasuk panggilan dari konsol Lambda MicroVMS dan panggilan kode ke operasi API MicroVMS Lambda.

Jika Anda membuat jejak, Anda dapat mengaktifkan pengiriman CloudTrail acara secara terus menerus ke bucket Amazon S3, termasuk peristiwa untuk Lambda MicroVMS. Jika Anda tidak mengonfigurasi jejak, Anda masih dapat melihat peristiwa terbaru di CloudTrail konsol di Riwayat acara.

Dengan menggunakan informasi yang dikumpulkan oleh CloudTrail, Anda dapat menentukan permintaan yang dibuat ke Lambda MicroVMS, alamat IP dari mana permintaan dibuat, siapa yang membuat permintaan, kapan itu dibuat, dan detail tambahan.

CloudTrail merekam dua jenis peristiwa untuk Lambda MicroVMS: peristiwa manajemen (operasi bidang kontrol seperti membuat dan menghapus gambar) dan peristiwa data (operasi bidang data

seperti menjalankan dan menghentikan microVM). Peristiwa manajemen dicatat secara default. Peristiwa data memerlukan opt-in eksplisit.

Peristiwa manajemen

Peristiwa manajemen MicroVMS Lambda adalah operasi bidang kontrol yang dicatat secara default. CloudTrail Acara manajemen termasuk membuat, memperbarui, dan menghapus gambar microVM, serta mencantumkan dan menjelaskan sumber daya.

Tindakan MicroVMS Lambda berikut dicatat sebagai peristiwa manajemen:

- `CreateMicrovmImage`
- `DeleteMicrovmImage`
- `DeleteMicrovmImageVersion`
- `UpdateMicrovmImage`
- `UpdateMicrovmImageVersion`
- `ListMicrovmImages`
- `GetMicrovmImage`
- `ListMicrovmImageVersions`
- `GetMicrovmImageVersion`
- `ListMicrovmImageBuilds`
- `GetMicrovmImageBuild`
- `ListMicrovms`
- `GetMicrovm`
- `ListManagedMicrovmImages`

Peristiwa data

Peristiwa data memberikan informasi tentang operasi sumber daya yang dilakukan pada atau di dalam sumber daya (misalnya, menjalankan atau menghentikan microVM). Ini juga dikenal sebagai operasi bidang data. Peristiwa data sering kali merupakan aktivitas bervolume tinggi. Secara default, CloudTrail tidak mencatat peristiwa data. Anda harus secara eksplisit mengaktifkan pencatatan peristiwa data untuk Lambda MicroVMS.

Jenis CloudTrail sumber daya untuk peristiwa data MicroVMS Lambda adalah. `AWS::Lambda::MicrovmImage` Anda dapat menggunakan CloudTrail konsol atau AWS CLI untuk mengonfigurasi jejak atau penyimpanan data peristiwa untuk mencatat peristiwa data.

Tindakan MicroVMS Lambda berikut dicatat sebagai peristiwa data:

- `RunMicrovm`
- `TerminateMicrovm`
- `SuspendMicrovm`
- `ResumeMicrovm`
- `CreateMicrovmAuthToken`
- `CreateMicrovmShellAuthToken`

Untuk mencatat peristiwa data untuk semua gambar microVM, gunakan pemilih peristiwa lanjutan berikut:

```
{
   "Name": "Log Lambda MicroVM data events",
   "FieldSelectors": [
     { "Field": "eventCategory", "Equals": ["Data"] },
     { "Field": "resources.type", "Equals": ["AWS::Lambda::MicrovmImage"] }
   ]
}
```

Mengaktifkan pencatatan peristiwa data

Untuk mengaktifkan pencatatan peristiwa data menggunakan AWS CLI, jalankan `put-event-selectors` perintah dengan pemilih acara lanjutan:

```
aws cloudtrail put-event-selectors \
   --trail-name my-trail \
   --advanced-event-selectors '[{
     "Name": "Log Lambda MicroVM data events",
     "FieldSelectors": [
       { "Field": "eventCategory", "Equals": ["Data"] },
       { "Field": "resources.type", "Equals": ["AWS::Lambda::MicrovmImage"] }
     ]
   }]'
```

Memahami entri CloudTrail log

Jej CloudTrail ak mengirimkan catatan aktivitas API ke bucket Amazon S3 yang Anda tentukan. Setiap entri log mewakili permintaan API tunggal dan mencakup identitas pemanggil, stempel waktu, parameter permintaan, dan respons.

Contoh berikut menunjukkan entri CloudTrail log yang menunjukkan peristiwa data untuk `RunMicrovm` tindakan:


```
{
  "eventVersion": "1.11",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "A1B2C3D4E5F6G7EXAMPLE:mySessionName",
    "arn": "arn:aws:sts::111122223333:assumed-role/MyRole/mySessionName",
    "accountId": "111122223333",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE"
  },
  "eventTime": "2026-04-20T18:26:44Z",
  "eventSource": "lambda.amazonaws.com",
  "eventName": "RunMicrovm",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "127.0.0.1",
  "requestParameters": {
    "microvmImageArn": "arn:aws:lambda:us-east-1:111122223333:microvm-image:my-image"
  },
  "responseElements": {
    "microvmId": "ai-12345678-abcd-1234-ef56-123456789abc",
    "microvmState": "PENDING",
    "endpoint": "12345678-abcd-1234-ef56-123456789abc.lambda-microvm.us-east-1.on.aws",
    "microvmImageArn": "arn:aws:lambda:us-east-1:111122223333:microvm-image:my-image",
    "microvmImageVersion": "1.0"
  },
  "requestID": "7aebcd0f-cda1-11e4-aaa2-e356da31e4ff",
  "eventID": "e92a3e85-8ecd-4d23-8074-843aabfe89bf",
  "readOnly": false,
  "resources": [
    {
      "accountId": "111122223333",
      "type": "AWS::Lambda::MicrovmImage",
      "ARN": "arn:aws:lambda:us-east-1:111122223333:microvm-image:my-image"
    }
  ],
```

```
  "eventType": "AwsApiCall",
  "managementEvent": false,
  "recipientAccountId": "111122223333",
  "eventCategory": "Data"
}
```

Pemecahan Masalah

Bagian ini menjelaskan cara men-debug dan memecahkan masalah umum saat bekerja dengan AWS Lambda MicroVM.

Akses shell

Gunakan akses shell untuk terhubung langsung ke microVM yang sedang berjalan untuk debugging dan pemecahan masalah.

Anda dapat terhubung ke shell microVM dengan dua cara:

- Konsol — Pilih MicroVM Anda di konsol Lambda dan pilih Connect.
- CLI — Hasilkan token shell dengan`create-microvm-shell-auth-token`, lalu gunakan token untuk membuat koneksi.

Buat token shell, lalu hubungkan:

```
aws lambda-microvms create-microvm-shell-auth-token \
  --microvm-identifier <id> --expiration-in-minutes 30
# In Console: select MicroVM -> Connect
# In shell: ctr task ls, then ctr task exec -t --exec-id shell <id> /bin/sh
```

MicroVM harus dijalankan dengan konektor `SHELL_INGRESS` jaringan (`arn:aws:lambda:us-east-1:aws:network-connector:aws-network-connector:SHELL_INGRESS`). Jika microVM tidak diluncurkan dengan konektor ini, `create-microvm-shell-auth-token` mengembalikan a`ValidationException`.

Untuk masalah lain:

- Periksa `terminationMessage` bidang dalam `get-microvm` respons untuk microVM yang dihentikan.

- Per CloudWatch iksa log build untuk masalah pembuatan gambar.
- Periksa `StateReason` bidang untuk konektor jaringan dalam `FAILED` keadaan.

Pemecahan masalah

Bagian ini memberikan solusi untuk masalah umum saat bekerja dengan Lambda MicroVMS.

Gejala	Kemungkinan penyebab dan resolusi
Pembuatan gambar gagal (`CREATION_FAILED`)	Periksa log build di`/aws/lambda-microvms/<image-name>`. Verifikasi `Dockerfile` sintaks, izin Amazon S3, dan ketersediaan gambar dasar. Jalan `docker build` kan secara lokal untuk mereproduksi.
MicroVM macet `PENDING`	Tunggu dan coba lagi. Jika persisten, periksa kesehatan layanan. Pastikan kuota konkurensi Anda tidak habis.
Aplikasi tidak merespons setelah dilanjutkan	Terapkan kait `/resume` siklus hidup untuk membangun kembali koneksi dan memvalidasi status. Periksa apakah aplikasi Anda terikat ke port 8080 (atau port yang dikonfigurasi) setelah dilanjutkan.
502 Bad Gateway dari titik akhir	Aplikasi macet atau tidak mendengarkan. Periksa log runtime. Veri `EXPOSE` fikasi dan `CMD` masuk`Dockerfile` . Jika melanjutkan otomatis, microVM mungkin gagal melanjutkan (periksa status dengan menggunakan`get-microvm`).
429 Terlalu Banyak Permintaan	Tingkat permintaan terlampaui. Coba lagi dengan backoff eksponensial dan jitter.
Koneksi terputus	Waktu tunggu idle dipicu. Terapkan ping/pong keepalives. Atau `maxIdleDurationSeconds` perpanjang kebijakan idle.
Latensi tinggi pada titik akhir	Saturasi bandwidth. Periksa apakah lalu lintas melebihi kemampuan bandwidth untuk ukuran microVM Anda. Skala hingga ukuran yang lebih besar.

Gejala	Kemungkinan penyebab dan resolusi
Token autentikasi kedaluwarsa (403)	Token memiliki kedaluwarsa yang dapat dikonfigurasi. Hasilkan token baru sebelum yang lama kedaluwarsa. Terapkan logika penyegaran token di klien Anda.
Keluar VPC tidak berfungsi	Verifikasi konektor jaringan dalam `ACTIVE` keadaan. Periksa aturan grup keamanan mengizinkan lalu lintas keluar. Konfirmasikan bahwa subnet memiliki rute ke sumber daya target Anda.

Kesalahan umum (pembuatan gambar)

Kesalahan	Penyebab	Solusi
`S3_ACCESS_DENIED`	Peran build tidak memiliki izin untuk mengambil artefak Amazon S3.	Tambahkan `s3:GetObject` izin untuk ember artefak Anda.
`S3_NO_SUCH_KEY`	Kunci artefak tidak ada di bucket.	Pastikan jalur Amazon S3 benar.
`S3_NO_SUCH_BUCKET`	Bucket Amazon S3 tidak ada.	Periksa nama bucket dan konfirmasikan telah dibuat.
`S3_INVALID_OBJECT`	Artefak di Glacier atau kelas penyimpanan yang tidak dapat diakses langsung.	Pindahkan artefak ke kelas penyimpanan Standar.
`S3_CROSS_REGION_ACCESS_DENIED`	Artefak berada di Wilayah yang berbeda dari gambar microVM.	Pastikan artefak Anda berada di Wilayah yang sama dengan gambar microVM Anda.
`ARCHIVE_DOCKERFILE_NOT_FOUND`	Arsip zip tidak ada `Dockerfile` di direktori root.	Tambahkan a `Dockerfile` ke root arsip zip Anda.

Kesalahan	Penyebab	Solusi
`ARCHIVE_INVALID`	File arsip bukan ZIP yang valid atau rusak.	Re-create arsip zip dan upload ulang.
`CONTAINER_BUILD_FAILED`	`Dockerfile` Instruksi tidak valid, file hilang, atau kesalahan sintaks.	Debug penggunaan `Dockerfile` `docker build` lokal Anda.
`DISK_STORAGE_FULL`	MicroVM kehabisan penyimpanan selama pembuatan.	Kurangi ukuran artefak atau hubungi dukungan.
`INTERNAL_PLATFORM_ERROR`	Terjadi kesalahan internal.	Coba lagi operasi. Jika gigih, hubungi dukungan.

Pemecahan masalah konektor jaringan

Kode kesalahan	Penyebab	Solusi
`DisallowedByVpcEncryptionControl`	VPC memiliki kebijakan kontrol enkripsi yang mencegah antarmuka jaringan atau lalu lintas yang tidak dienkripsi. Lambda tidak dapat membuat ENI yang memenuhi persyaratan enkripsi.	Tambahkan Lambda ke daftar pengecualian kontrol enkripsi VPC. Jika pengecualian tidak memungkinkan, gunakan VPC atau subnet yang tidak memiliki kontrol enkripsi ketat yang diterapkan.
`Ec2RequestLimitExceeded`	Lambda membuat panggilan API EC2 (misalnya,`CreateNetworkInterface` ,`DescribeSubnets`) untuk mengatur konektivitas. Terlalu banyak panggilan API EC2 bersamaan menyebabkan pelambatan.	Coba kembali operasi setelah penundaan singkat. Jika persisten , kurangi operasi konektor jaringan bersamaan atau minta peningkatan batas pelambatan API EC2 melalui Dukungan. AWS

Kode kesalahan	Penyebab	Solusi
`InsufficientRolePermissions`	Peran operator tidak memiliki izin EC2 yang diperlukan.	Pastikan peran IAM memiliki izin jaringan EC2 yang diperlukan.
`InternalError`	Kesalahan tak terduga terjadi dalam layanan Lambda saat memproses permintaan konektor jaringan.	Coba lagi operasi. Jika terus berlanjut setelah beberapa kali percobaan ulang, hubungi AWS Dukungan dengan konektor jaringan ARN dan perkiraan stempel waktu.
`InvalidSecurityGroup`	ID grup keamanan tidak ada, telah dihapus, atau bukan milik VPC yang sama dengan subnet yang ditentukan.	Verifikasi semua ID grup keamanan ada dan milik VPC yang sama dengan subnet. Gunakan `aws ec2 describe-security-groups --group-ids <sg-id>` untuk memvalidasi.
`InvalidSubnet`	ID subnet tidak ada, telah dihapus, atau milik VPC yang berbeda dari yang diharapkan.	Verifikasi semua ID subnet ada dan milik VPC yang benar. Gunakan `aws ec2 describe-subnets --subnet-ids <subnet-id>` untuk memvalidasi.
`SubnetOutOfIPAddresses`	Blok CIDR subnet habis — semua IP dialokasikan ke ENI, instance, dan sumber daya lainnya, sehingga Lambda tidak dapat membuat antarmuka jaringan.	Kosongkan alamat IP dengan menghapus yang tidak digunakan ENIs/instances, atau gunakan subnet yang berbeda dengan kapasitas yang tersedia. Pertimbangkan subnet yang lebih besar (misalnya, /24 atau lebih besar) untuk konektor jaringan.

Praktik terbaik untuk Lambda MicroVMS

Ikuti praktik terbaik ini untuk membangun aplikasi yang andal, hemat biaya, dan aman dengan AWS Lambda MicroVMS.

Desain aplikasi

Rancang aplikasi Anda agar bekerja dengan baik dengan snapshot microVM dan kait siklus hidup:

- Gunakan kait siklus hidup — Terapkan `/run` untuk inisialisasi pasca-proses, `/suspend` untuk pembersihan yang anggun, untuk pembentukan kembali koneksi, dan `/resume` `/terminate` untuk pembilasan data.
- Menangani penggunaan kembali snapshot - Hasilkan UUID, rahasia, dan nilai acak di `/run` hook, bukan selama pembuatan gambar. Gunakan CSPRNGs untuk semua pembuatan nomor acak.
- Desain untuk resume - Validasi semua koneksi jaringan dan status cache di `/resume` hook. AWS Koneksi SDK biasanya pulih secara otomatis, tetapi koneksi khusus mungkin tidak.

Performa

Optimalkan kinerja microVM dengan praktik berikut:

- Right-size MicroVM Anda — Pilih ukuran microVM terkecil yang memenuhi persyaratan CPU, memori, dan bandwidth Anda. Skala bandwidth dengan memori yang dikonfigurasi.
- Minimalkan ukuran snapshot — Jaga agar jejak aplikasi Anda tetap kecil untuk mengurangi waktu berjalan dan melanjutkan. Hapus dependensi waktu pembuatan dan file sementara sebelum snapshot gambar diambil.
- Gunakan suspend/resume untuk beban kerja yang sensitif terhadap latensi — Pertahankan lingkungan yang sering digunakan dalam keadaan ditangguhkan untuk melanjutkan eksekusi dengan cepat.

Keamanan

Ikuti praktik berikut untuk mengamankan beban kerja MicroVM Anda:

- Putar token autentikasi — Hasilkan token berumur pendek (15-30 menit). Terapkan logika penyegaran token di klien Anda sebelum kedaluwarsa.
- Token cakupan ke port tertentu — Gunakan `allowedPorts` parameter untuk membatasi cakupan token hanya pada port yang digunakan aplikasi Anda.
- Gunakan keluar VPC untuk lalu lintas sensitif — Rute lalu lintas ke database dan API internal melalui konektor keluar VPC daripada internet publik.

- Terapkan IAM hak istimewa terkecil - Pisahkan peran build (pembuatan gambar) dari peran eksekusi (runtime). Berikan izin minimum yang diperlukan.

Optimalisasi biaya

Kurangi biaya dengan strategi berikut:

- Konfigurasikan kebijakan idle — Setel `maxIdleDurationSeconds` untuk menangguhkan microVM idle secara otomatis. MicroVM yang ditangguhkan tidak dikenakan biaya komputasi.
- Tetapkan maksimum DurationInSeconds — Cegah biaya tak terduga dengan mengatur waktu penghentian yang sulit pada setiap microVM.
- Hentikan microVM yang tidak digunakan — Jangan biarkan microVM berjalan tanpa batas waktu. Gunakan `suspendedDurationSeconds` untuk mengakhiri secara otomatis setelah periode idle yang diperpanjang.
- Sumber daya tag — Gunakan tag pada gambar MicroVM untuk alokasi biaya dan pelaporan di Co AWS st Explorer.
- Right-size MicroVM Anda — Mulailah dengan baseline terkecil yang memenuhi kebutuhan kinerja Anda. Anda dapat melakukan burst hingga 4x baseline selama aktivitas puncak tanpa penyediaan ulang.
- Bersihkan gambar yang tidak digunakan — Hapus versi gambar MicroVM yang tidak lagi Anda butuhkan untuk mengurangi biaya penyimpanan.

Memantau

Pantau MicroVM Anda secara efektif:

- Tinjau StateReason - Ketika MicroVMS berhenti secara tidak terduga, `get-microvm` respons menyertakan alasan kegagalan yang `stateReason` tidak terduga.

Integrasi

AWS Lambda MicroVMS menyediakan lingkungan komputasi berbasis snapshot yang terisolasi yang dapat digunakan oleh kerangka kerja agen AI dan alat pengembangan sebagai kotak pasir. Setiap integrasi mengikuti pola yang sama: Anda membuat gambar microVM yang dapat digunakan kembali, dan kerangka kerja meluncurkan satu microVM per sesi atau pekerjaan melalui Lambda API.

Integrasi berikut tersedia:

- the section called “Agen Terkelola Claude”— Gunakan Lambda MicroVMS sebagai kotak pasir yang dihosting sendiri untuk Agen Terkelola Claude Anthropic. Loop agen berjalan pada infrastruktur Anthropic; microVM menjalankan panggilan alat Anda. Pengontrol berbasis webhook meluncurkan satu microVM per sesi.
- the section called “Agen Cloud Cursor”— Gunakan Lambda MicroVMS sebagai mesin yang di-host sendiri untuk Cursor Cloud Agents. Pengontrol terjadwal melakukan polling untuk permintaan kumpulan yang tertunda dan meluncurkan microVM untuk setiap sesi pekerja.
- the section called “API Agen OpenAI”— Gunakan Lambda MicroVMS sebagai lingkungan yang dihosting sendiri untuk agen yang dibangun dengan OpenAI Agents API. Aplikasi Anda atau peluncur webhook memulai microVM untuk setiap belokan, dengan penangguhan opsional dan lanjutkan di antara belokan.

Menggunakan Lambda MicroVMS sebagai kotak pasir untuk Claude Managed Agents

AWS Lambda MicroVMS adalah penyedia sandbox terkelola di kotak pasir yang di-host sendiri untuk Claude Managed Agents; menjaga file, paket, dan layanan sensitif dalam infrastruktur yang Anda kendalikan. Anthropic menghosting loop agen dan model Claude, sedangkan Lambda microVM adalah tempat panggilan alat Anda berjalan. Dengan pola ini, Anda mengontrol lingkungan eksekusi — apa yang diinstal, akses jaringan apa yang tersedia, dan sumber daya apa yang dapat dijangkau agen.

Setiap microVM adalah mesin Firecracker-isolated virtual dengan startup sub-detik berbasis snapshot, berjalan hingga 8 jam, dan dapat dihentikan ketika sesi berakhir. Sesi tidak pernah berbagi status. Anda mendapatkan batas keamanan VM dengan model operasional tanpa server — tidak ada cluster untuk dikelola, tidak ada kapasitas idle yang harus dibayar. Anda dapat mengatur microVM dengan AWS layanan untuk mengimplementasikan spesifikasi sandbox yang di-host sendiri seperti yang dijelaskan di bawah ini. Tinjau Self-Hosted kotak pasir Claude Managed Agents pada sampel referensi MicroVMS Lambda untuk detail lebih lanjut.

Cara kerjanya

Pesawat kontrol meluncurkan satu microVM per sesi Claude:

1. Sesi mencapai status berjalan dan Anthropic mengirimkan `session.status_run_started` webhook ke titik akhir di akun Anda.
2. Fungsi peluncur Lambda memverifikasi tanda tangan webhook, lalu memanggil. `RunMicroVM`
3. Kode Anda di MicroVM mengklaim sesi, mengeksekusi panggilan alat (bash, baca, tulis, edit, glob, grep) di`/workspace`, dan memposting hasil kembali ke Anthropic.
4. MicroVM ditangguhkan atau dihentikan ketika sesi berakhir.

Kunci API organisasi Anthropic Anda tidak pernah mencapai AWS komputasi. Peluncur hanya meneruskan referensi Manajer AWS Rahasia ke kunci lingkungan Anthropic Anda; peran eksekusi MicroVM memungkinkan kode untuk membacanya saat runtime.

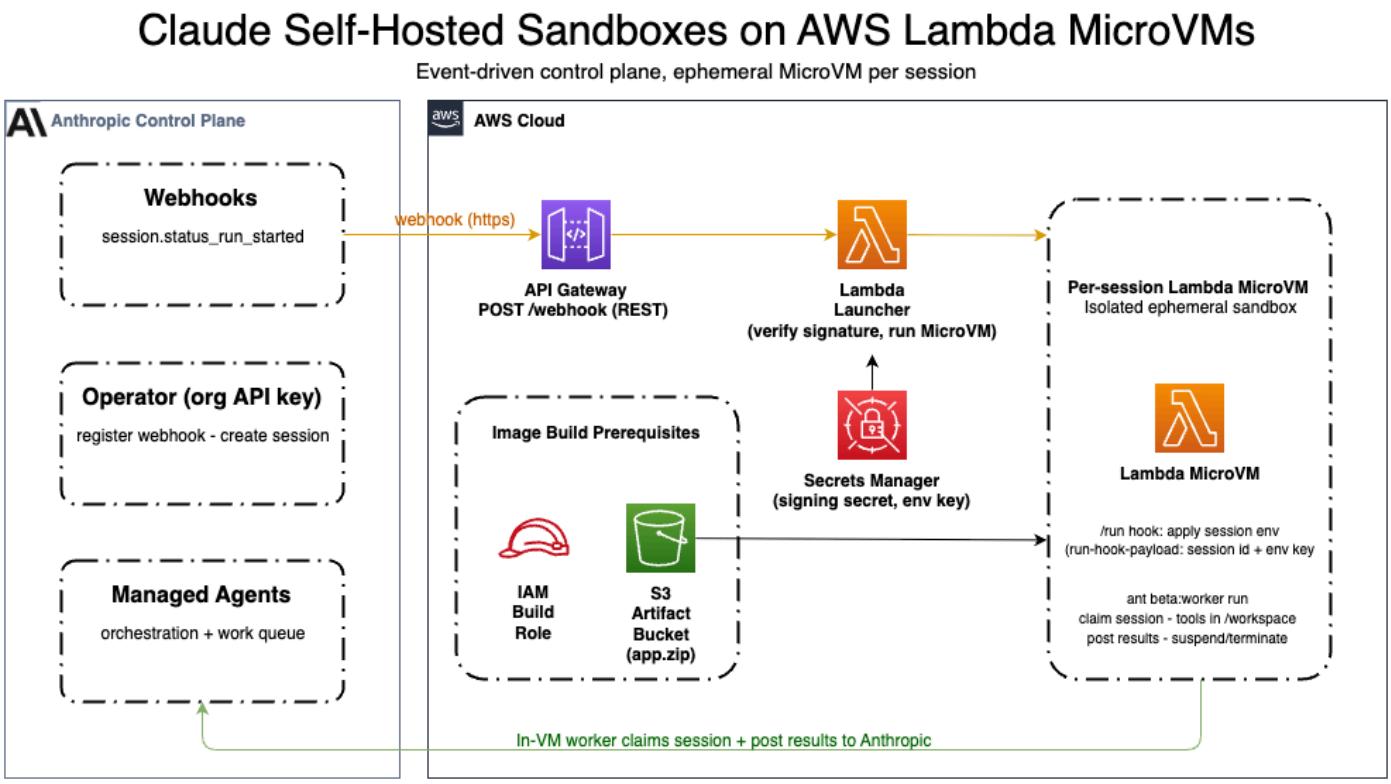
Properti utama

Properti	Manfaat
Isolasi petasan	Hardware-virtualized batas per sesi
Snapshot-based sepatu bot	Lanjutkan dari snapshot Firecracker dalam sub-detik hingga detik satu digit
IAM melalui IMDSv2	Menggunakan kredentif jangka pendek dan hak istimewa terendah
Durasi stateful	Dapat berjalan hingga 8 jam dengan akses disk dan memori penuh
Pay-per-session	MicroVM dihentikan saat sesi selesai, yang mengakhiri penagihan

Prasyarat

- AWS Akun dengan izin untuk Amazon Simple Storage Service (Amazon S3), AWS Identity and Access Management (IAM), Secrets Manager,, AWS WAF, Lambda, dan MicroVMS Lambda
- Agen Agen Terkelola Claude dengan `self_hosted` lingkungan
- Kunci lingkungan dan rahasia penandatanganan webhook dari Konsol Claude

Menyebarkan implementasi referensi



Kotak Pasir Agents Ter Self-Hosted kelola Claude pada repositori MicroVMS Lambda menyediakan penerapan yang minimal dan berfungsi. Ini termasuk:

- Tumpu CloudFormation kan (Amazon API Gateway, fungsi Lambda peluncur, rahasia Manajer Rahasia, bucket Amazon S3, peran IAM)
- Gambar microVM (`Dockerfile`, Node.js EnvironmentWorker, kait siklus hidup)
- Skrip penerapan dan skrip verifikasi

Langkah-langkah penyebaran:

1. Terapkan tum CloudFormation pukan.
2. Simpan kunci lingkungan dan penandatanganan rahasia di rahasia Manajer Rahasia yang dibuat.
3. Membangun gambar microVM.
4. Daftarkan URL webhook tumpukan di Konsol Claude.
5. Verifikasi dengan membuat sesi.

Untuk petunjuk terperinci, lihat repositori README.

Jaringan

Lambda MicroVM memiliki akses internet publik secara default - tidak diperlukan konfigurasi untuk menjangkau. `api.anthropic.com`

Untuk mengakses sumber daya pribadi seperti database Amazon Aurora atau ElastiCache cluster Amazon, atau untuk menerapkan batasan jaringan Anda sendiri, pasang konektor keluar VPC pada waktu peluncuran. Lihat Bekerja dengan konektor jaringan jalan keluar.

Kebijakan idle

Setel `suspendedDurationSeconds: 0` dan `autoResumeEnabled: false` untuk microVM per sesi. Tetapkan `maximumDurationInSeconds` sebagai plafon untuk sesi macet (maks 28.800 detik).

```
{
  "maxIdleDurationSeconds": 120,
  "suspendedDurationSeconds": 0,
  "autoResumeEnabled": false
}
```

Memantau

Log aplikasi - Tinjau log aplikasi di CloudWatch Log:

```
aws logs tail /aws/lambda-microvms/claude-worker --follow
```

Menjalankan MicroVMS — Daftar microVM yang sedang berjalan di akun Anda:

```
aws lambda-microvms list-microvms \
  --image-identifier claude-worker \
  --query 'items[].[microvmId,state,startedAt]' --output table
```

Pemecahan masalah

Gejala	Penyebab
Webhook mengembalikan 401	Menandatangani ketidakcocokan rahasia atau pengiriman basi

Gejala	Penyebab
Tidak ada peluncuran MicroVM	Webhook tidak terdaftar`session.status_run_started` , atau peluncur peran eksekusi fungsi Lambda kehilangan izin `RunMicroVM`
MicroVM segera berakhir	`/run`batas waktu kait - naikkan `runTimeoutInSeconds`
Pekerja segera keluar	HTTPS keluar untuk diblokir `api.anthropic.com`
Pembuatan gambar gagal `S3_*`	Membangun kesalahan konfigurasi peran atau bucket

Menggunakan Lambda MicroVMS sebagai kotak pasir untuk Agen Cloud Cursor

Anda dapat menggunakan AWS Lambda MicroVM sebagai mesin yang di-host sendiri untuk Agen Cloud Cursor Anda, menyimpan repositori, eksekusi alat, dan akses jaringan di infrastruktur yang Anda kendalikan. Kursor menghosting loop dan model agen; Lambda microVM adalah tempat pekerja mengklaim permintaan kumpulan dan menjalankan panggilan alat. Dengan pola ini, Anda mengontrol lingkungan eksekusi — apa yang diinstal dalam gambar, akses jaringan apa yang tersedia, dan AWS sumber daya apa yang dapat dijangkau oleh peran IAM pekerja.

Setiap MicroVM adalah mesin Firecracker-isolated virtual dengan startup berbasis snapshot, berjalan hingga 8 jam, dan dihentikan ketika sesi berakhir. Sesi tidak pernah berbagi status. Anda mendapatkan batas keamanan VM dengan model operasional tanpa server — tidak ada cluster untuk dikelola, tidak ada kapasitas kumpulan idle yang harus dibayar. Anda mengatur microVM dengan fungsi Lambda pengontrol terjadwal yang menanggapi permintaan kumpulan yang tertunda. Tinjau template referensi Jalankan mesin yang di-host sendiri untuk Agen Cloud Cursor di AWS Lambda MicroVMS untuk detail selengkapnya.

Cara kerjanya

Pengontrol terjadwal meluncurkan satu microVM per permintaan kumpulan yang tertunda:

1. Anda memulai agen cloud di kursor. com/agentsterhadap mesin yang di-host sendiri. Permintaan tetap tertunda sampai pekerja mengklaimnya.

2. Fungsi Lambda controller berjalan `agent worker controller --spawn ./spawn.sh` selama jendela polling Ev Server-Sent ents (SSE) lima menit. Ketika melihat permintaan yang cocok, ia berjalan`spawn.sh`.
3. `spawn.sh`memang `RunMicrovm` gil (`aws lambda-microvms run-microvm`) dan kembali tanpa menunggu. Ini meneruskan `CURSOR_*` lingkungan klaim ke tamu melalui`--run-hook-payload`.
4. Pengait microVM `/run` (`hook.py`) menerapkan payload dan mulai`entrypoint.sh`, yang berjalan. `cursor-agent worker --pool … start` Pekerja mengeksekusi panggilan alat di akun Anda.
5. Ketika sesi tidak aktif melewati batas waktu rilis, pekerja melepaskan dan microVM dihentikan.

Kunci API akun layanan Cursor Anda disimpan di Penyimpanan Parameter Manajer AWS Sistem sebagai. `SecureString` Pengontrol dan microVM membacanya saat runtime`ssm:GetParameter`; kunci tidak pernah dimasukkan ke dalam gambar.

Properti utama

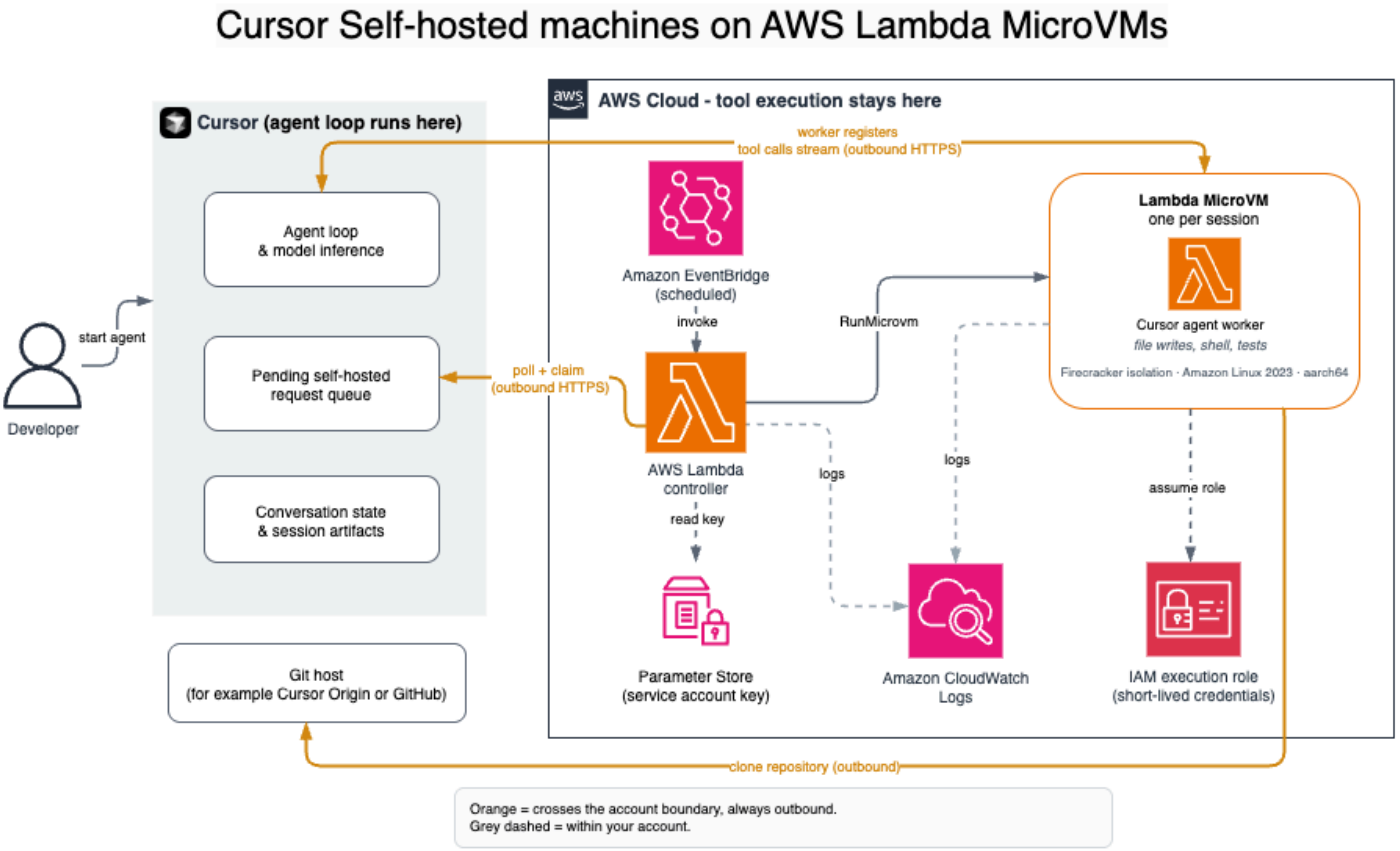
Properti	Manfaat
Isolasi petasan	Hardware-virtualized batas per sesi
Boot snapshot	Lambda mengembalikan tamu dari snapshot Firecracker gambar Anda
IAM melalui peran eksekusi	Tamu menggunakan kredensi berumur pendek dari `MicroVmExecutionRoleArn`
Durasi stateful	Setiap microVM dapat berjalan hingga 8 jam () `--maximum-duration-in-seconds 28800`
Pay-per-session	Anda membayar untuk runtime microVM, bukan untuk kapasitas kumpulan idle

Prasyarat

- AWS Akun dengan Lambda MicroVMS diaktifkan, ditambah izin untuk menggunakan Amazon S3, IAM CloudFormation, dan Penyimpanan Parameter Manajer Sistem

- Akun Cursor Enterprise
- Tim Cursor dengan mesin yang di-host sendiri diaktifkan
- K unci API akun layanan untuk pekerja kumpulan
- AWS CLI diperbarui ke versi stabil terbaru, dan Docker (skrip penerapan membangun image pengontrol)

Menyebarkan implementasi referensi



Repositori anysphere/aws -lambda-workers menyediakan penerapan minimal yang berfungsi. Ini termasuk:

- CloudFormation Stack (`cloudformation.yaml`): fungsi Lambda pengontrol terjadwal, EventBridge `rate(1 minute)` aturan, bucket artefak Amazon S3, antrian huruf mati, dan peran IAM (,,) `MicroVmExecutionRole` `BuildRole` `SpawnRole` `ControllerRole`
- Gambar pengontrol (`controller/Dockerfile`,`controller/handler.py`) yang menjalankan satu jendela polling SSE per pemanggilan
- Sebuah gambar microVM (`microvm-image/`:`Dockerfile`,`hook.py`,`entrypoint.sh`)

- Sebuah spawn hook (`spawn.sh`) dan script penerapan (`deploy.sh`)

Langkah-langkah penyebaran:

1. Simpan kunci API akun layanan di SSM.

```
aws ssm put-parameter --type SecureString \
  --name /cursor-lambda-workers/cursor-api-key \
  --value "YOUR_SERVICE_ACCOUNT_KEY"
```

2. Terapkan tumpukan. `deploy.sh`membangun gambar pengontrol dan menjalankan`aws cloudformation deploy`.

```
POOL_NAMES=default ./deploy.sh
# POOL_NAMES=gpu,default ./deploy.sh
# ALL_POOLS=true ./deploy.sh
```

3. Bangun gambar microVM dari `microvm-image/` menggunakan output tumpukan. Aktif `ready` kan kait `validate` gambar dan kait `run` microVM sehingga mereka ditangkap di jalur snapshot.

```
BUCKET=$(aws cloudformation describe-stacks --stack-name cursor-lambda-workers \
  --query "Stacks[0].Outputs[?OutputKey=='ArtifactBucketName'].OutputValue" --output
 text)
BUILD_ROLE=$(aws cloudformation describe-stacks --stack-name cursor-lambda-workers \
  --query "Stacks[0].Outputs[?OutputKey=='BuildRoleArn'].OutputValue" --output text)
BASE=$(aws lambda-microvms list-managed-microvm-images --query "items[0].imageArn" --
output text)
( cd microvm-image && zip -r /tmp/app.zip . )
aws s3 cp /tmp/app.zip "s3://${BUCKET}/app.zip"
aws lambda-microvms create-microvm-image \
  --code-artifact "uri=s3://${BUCKET}/app.zip" \
  --name cursor-pool-worker \
  --base-image-arn "${BASE}" \
  --build-role-arn "${BUILD_ROLE}" \
  --environment-variables "POOL_NAME=default,CURSOR_API_KEY_PARAM_NAME=/cursor-
lambda-workers/cursor-api-key" \
  --hooks '{"port":9000,"microvmImageHooks":
{"ready":"ENABLED","readyTimeoutInSeconds":60,"validate":"ENABLED","validateTimeoutInSeconds"
{"run":"ENABLED","runTimeoutInSeconds":60}}'
```

4. Mulai agen cloud dari kursor. com/agentsterhadap kumpulan (atau terhadap repositori dalam mode repo-bound) untuk memverifikasi.

Untuk petunjuk terperinci, lihat repositori README.

Mode kolam dan repo

Jaga agar tumpukan `PoolNames` dan gambar `POOL_NAME` sejajar sehingga pengontrol dan tamu melayani kolam yang sama.

- Any-repo mode — Perutean berdasarkan nama kumpulan, bukan oleh git remote. Tamu memulai pekerja dari ruang kerja tanpa git remote, sehingga pekerja menghilangkan `repo=` label. Pengguna memilih grup repo apa saja dan nama kumpulan saat memulai agen.
- Repo-bound mode — Pekerja melayani satu atau lebih remote git tertentu. Baik memanggang klon ke dalam gambar microVM, atau atur `CURSOR_REPO_URL` sehingga meng `entrypoint.sh` kloningnya di awal. Pekerja memperoleh `repo=` label dari jarak jauh git. Repositori pribadi memerlukan kredenSIAL git pada pekerja.

Jaringan

Referensi `spawn.sh` melampirkan `INTERNET_EGRESS` konektor terkelola sehingga pekerja dapat mencapai bidang kontrol Cursor, dan `ALL_INGRESS` untuk kait gambar. Outbound-only pekerja tidak pernah menerima HTTPS masuk setelah boot.

Untuk menjangkau sumber daya pribadi (misalnya database Amazon Aurora atau ElastiCache cluster Amazon) atau untuk menerapkan batasan Anda sendiri, pasang konektor keluar VPC pada waktu peluncuran sebagai pengganti. `INTERNET_EGRESS` Lihat the section called “Bekerja dengan konektor jaringan jalan keluar”.

Memantau

Log tamu - log MicroVM masuk ke CloudWatch Log:

```
aws logs tail /aws/lambda/microvms/cursor-pool-worker --follow
```

Log pengontrol — Pengontrol fungsi Lambda mencatat ke`/aws/lambda/cursor-lambda-workers-controller`.

Menjalankan MicroVMS — Daftar microVM yang berjalan untuk gambar:

```
aws lambda-microvms list-microvms --image-identifier cursor-pool-worker
```

Pemecahan masalah

Gejala	Penyebab
Pembuatan gambar gagal (S3 atau IAM)	Verifikasi output tumpukan `ArtifactBucketName` dan`BuildRoleArn` ; zip harus mendarat di bucket itu dan peran build harus dapat membacanya
Tidak ada peluncuran MicroVM	Pengontrol tidak berjalan atau tidak dapat memang `RunMicrovm` gil; konfir `cursor-pool-worker` masikan gambar ada dan (untuk pengontrol lokal) yang `SpawnRoleArn` diasumsikan
Pekerja segera keluar	Tamu hilang `CURSOR_API_KEY` (SSM`/cursor-lambda-workers/cursor-api-key`), atau `/run` kait tidak dimulai `cursor-agent worker … start`
`Exec format error` pada node	Gambar menginstal arsitektur CLI yang salah; MicroVMS di sini adalah aarch64 () `arm64`
Service-account kunci dilaporkan tidak valid	`CURSOR_API_ENDPOINT` /`CURSOR_API_URL` diteruskan ke tamu; ini harus tetap tidak disetel sehingga `worker start` menggunakan host autentikasi default
Gambar dibuat sebelum kait diaktifkan	Bangun kembali gambar microVM sehinggaready,`validate`, dan `run` kait berada di jalur snapshot

Sumber daya terkait

- AWS MicroVM Lambda
- RunMicrovmReferensi CLI
- Mulai cepat mesin yang di-host sendiri kursor
- Templat referensi: anysphere/aws -lambda-worker

Menggunakan Lambda MicroVMS sebagai kotak pasir untuk OpenAI Agents API

Anda dapat menggunakan AWS Lambda MicroVMS sebagai lingkungan yang di-host sendiri untuk agen yang dibangun dengan OpenAI Agents API, menyimpan file ruang kerja, eksekusi alat, dan akses jaringan di infrastruktur yang Anda kendalikan. OpenAI menjalankan harness agen; Lambda microVM mengeksekusi tugas Anda. Dalam contoh ini, ia berjalan`codex exec-server`, yang mengeksekusi panggilan alat agen dan menyimpan file ruang kerja sesi. Anda memutuskan konten gambar, jalur jaringan yang dapat diambil pelaksana, dan AWS sumber daya yang menjadi cakupan peran pelaksanaannya.

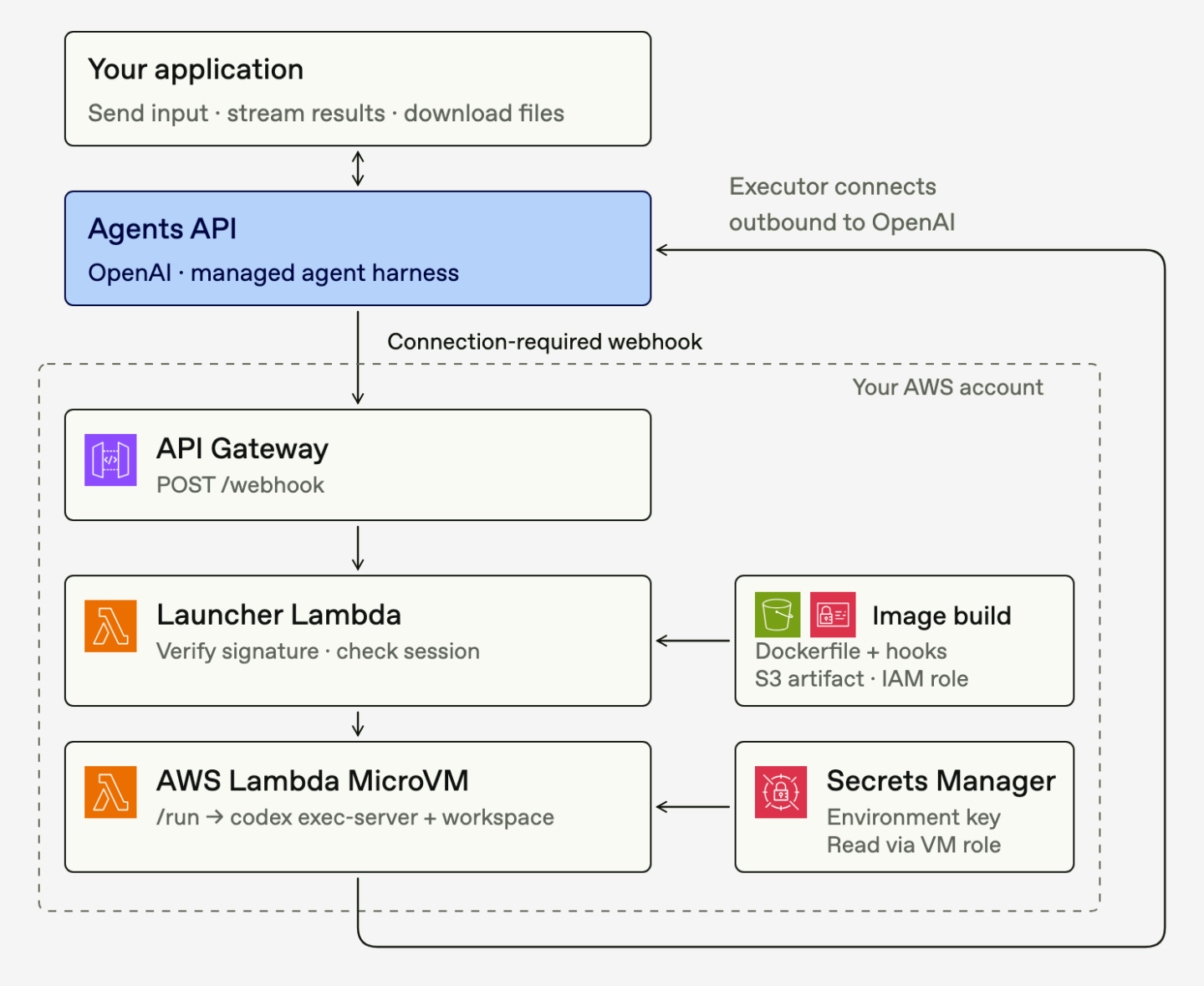
Setiap microVM berjalan sebagai Firecracker-isolated lingkungan yang melakukan boot dari snapshot gambar Anda. Total masa pakai microVM hingga 8 jam, termasuk waktu berjalan dan ditangguhkan. Hentikan setelah belokan terakhir, atau tunda di antara belokan. Anda membuat satu gambar yang dapat digunakan kembali, lalu meluncurkan microVM dari aplikasi Anda atau peluncur webhook. Kedua pendekatan menggunakan gambar dan kredenSIAL yang sama. Untuk detail selengkapnya, tinjau contoh referensi untuk kotak pasir yang di-host sendiri OpenAI Agents API di Lambda MicroVMS.

Cara kerjanya

Di jalur yang dikelola webhook, fungsi peluncur Lambda memulai microVM ketika sesi pertama kali membutuhkan eksekutor. Ini menggunakan kembali microVM yang ada, melanjutkannya terlebih dahulu jika ditangguhkan. Aplikasi Anda mengakhirinya setelah giliran terakhir. Gambar 1 menunjukkan aliran:

1. Aplikasi Anda membuat sesi yang di-host sendiri dan mengirimkan masukan ke API Agen.
2. Saat sesi membutuhkan eksekutor, OpenAI mengirimkan `agent.session.action_required` webhook dengan `environment_connection` tindakan ke titik akhir Amazon API Gateway di akun Anda.
3. Fungsi peluncur Lambda memverifikasi tanda tangan webhook, mengambil sesi saat ini, dan mengonfirmasi bahwa handler memilikinya.
4. Peluncur memanggil `RunMicrovm` dengan gambar, peran eksekusi, konektor jaringan, dan `runHookPayload` yang membawa nilai koneksi sesi.
5. Pengait MicroVM menggunakan peran eksekusi IAM untuk mengambil kunci eksekutor dari AWS Secrets Manager dan memulai Codex. `/run`

6. Pelaksana mendaftar dengan OpenAI di. `https://api.openai.com` Kemudian membuka WebSocket koneksi aman `wss://codex-cloud-environments.chatgpt.com` untuk menerima perintah alat dan mengembalikan hasil.
7. Aplikasi Anda dapat mengunggah file input dan mengunduh file output melalui titik akhir HTTPS yang diautentikasi MicroVM. Ini mengikuti aliran acara sesi dan dapat menangguhkan microVM di antara belokan.
8. Setelah giliran terakhir, aplikasi Anda mengakhiri microVM.



Gambar 1. Webhook-managed penyediaan untuk kotak pasir API Agents di MicroVMS AWS Lambda.

Jaga rahasia kunci aplikasi dan penandatanganan webhook di luar microVM. Peluncur hanya melewati ARN rahasia kunci eksekusi; VM mengambil kunci menggunakan peran eksekusi.

Properti utama

Properti	Manfaat
Isolasi petasan	Hardware-virtualized batas untuk microVM setiap sesi
Snapshot-based startup	Setiap microVM diluncurkan dari snapshot gambar yang Anda siapkan
IAM melalui peran eksekusi	Peng `/run` ait membaca kunci eksekutor dengan peran eksekusi microVM; kunci tidak disimpan dalam gambar
Durasi status	Setiap microVM dapat berjalan hingga 8 jam, termasuk waktu yang ditangguhkan
Tangguhkan dan lanjutkan di antara belokan	Memori dan status disk dipertahankan sehingga putaran tindak lanjut berlanjut di lingkungan yang sama
Usage-based menghitung	Menjalankan VM dikenakan biaya komputasi. Penangguhan menghentikan biaya komputasi; penyimpanan snapshot dan reads/writes ditagih secara terpisah.

Prasyarat

- AWS Akun dengan akses Lambda MicroVMS, versi AWS CLI yang menyertakan`lambda-microvms`, dan izin untuk membuat gambar dan menjalankan MicroVMS. Untuk membuat bucket artefak Amazon Simple Storage Service (Amazon S3) dan peran pembuatan gambar AWS Identity and Access Management (IAM), lihat. the section called “Buat MicroVM pertama Anda”
- Kunci aplikasi OpenAI (`OPENAI_API_KEY`) untuk membuat sesi dan mengirimkan masukan.
- Kunci eksekutor OpenAI terpisah (`OPENAI_EXECUTOR_API_KEY`) dengan organisasi, proyek, dan akun pengguna atau layanan yang sama dengan sesi. Simpan di Secrets Manager, misalnya sebagai`codex/agents-api/executor`. Peng `/run` ait meneruskannya ke Codex sebagai`CODEX_API_KEY`.

- Peran eksekusi MicroVM yang hanya dapat membaca rahasia kunci eksekutor. Jika rahasia menggunakan AWS kunci Key Management Service (AWS AWS KMS) yang dikelola pelanggan, tambahkan izin dekripsi. Lihat the section called “Keamanan”.

Mempersiapkan gambar microVM

Buat gambar yang menyertakan Codex CLI, dependensi alat Anda, direktori kerja seperti `/workspace`, dan server HTTP untuk kait siklus hidup.

Kait	Perilaku
`/aws/lambda-microvms/runtime/v1/ready`	Mengembalikan HTTP 200 ketika server siap untuk Lambda mengambil snapshot. Pelaksana tetap terputus.
`/aws/lambda-microvms/runtime/v1/run`	Membaca nilai koneksi sesi dan ARN rahasia, mengambil kunci eksekutor, dan memulai eksekutor.
`/aws/lambda-microvms/runtime/v1/validate`	Opsional. Memeriksa apakah Codex berjalan dan ruang kerja dapat ditulis.
`/aws/lambda-microvms/runtime/v1/suspend`	Diperlukan untuk suspend/resume. Menghentikan eksekutor dan membilas penulisan sebelum mengambil AWS snapshot VM.
`/aws/lambda-microvms/runtime/v1/resume`	Diperlukan untuk suspend/resume. Mengambil kunci lagi dan memulai ulang eksekutor untuk lingkungan yang sama.
`/aws/lambda-microvms/runtime/v1/terminate`	Opsional. Menghentikan eksekutor dan membilas tulisan sebelum penghentian.

Kemas `Dockerfile` dan kaitkan server dalam artefak Amazon S3, lalu buat `codex-executor` gambar dengan memori dasar 8 GB (4 dasar vCPU). Aktifkan kait yang dipilih dalam konfigurasi gambar dan atur port kait agar sesuai dengan server Anda. Jauhkan ID sesi, kredenSIAL, dan

koneksi eksekutor langsung dari snapshot gambar. Untuk konfigurasi build dan hook, lihatthe section called “Image MicroVM”.

Saat diluncurkan, aplikasi atau peluncur Anda membuat serial nilai-nilai berikut sebagai JSON di: `runHookPayload`

Nilai	Tujuan
`session.environment.id`	Mengidentifikasi lingkungan yang terhubung dengan pelaksana
`session.environment.remote_url`	Menyediakan URL koneksi OpenAI pelaksana
Executor-key rahasia ARN	Memungkinkan hook mengambil kunci menggunakan peran eksekusi microVM

Kembali dari `/run` hook setelah proses eksekutor dimulai, tanpa menunggu giliran selesai, untuk menghindari batas waktu hook.

Meluncurkan MicroVMS

Anda dapat meluncurkan MicroVMS dari aplikasi Anda atau dari peluncur webhook. Dalam kedua kasus tersebut, biarkan aliran acara sesi tetap terbuka sampai giliran selesai.

Application-managed penyediaan

1. Buat sesi yang di-host sendiri yang direktori kerjanya cocok dengan gambar, lalu buka aliran acaranya.
2. Panggil `RunMicrovm` dengan gambar ARN dan versi, peran eksekusi, konektor jaringan, dan`runHookPayload`. Simpan yang dikembalikan di `microvmId` samping ID sesi.
3. Tunggu`agent.session.environment.connected`, lalu kirim input dan ikuti hasil giliran.
4. Ambil file output dan akhiri giliran seperti yang dijelaskan dalamthe section called “Mengakhiri giliran”.

Webhook-managed penyediaan

Aplikasi Anda membuat sesi, membuka streamnya, dan mengirimkan masukan. Peluncur menangani sisanya saat webhook tiba.

1. Terapkan `POST /webhook` rute di Amazon API Gateway yang memanggil peluncur. Berikan izin peluncur untuk membaca kredensialnya, memeriksa, meluncurkan, melanjutkan, dan menghentikan microVM dari gambar yang dipilih, meneruskan peran eksekusi microVM, dan menggunakan konektor jaringan yang dikonfigurasi.
2. Daftarkan titik akhir untuk `agent.session.action_required` dan`agent.session.failed`. Simpan rahasia penandatanganan titik akhir di Secrets Manager dengan kunci aplikasi peluncur.
3. Verifikasi tanda tangan webhook terhadap badan permintaan mentah sebelum mengakses sesi atau meluncurkan komputasi. Kemudian ambil sesi saat ini dan konfirmasikan bahwa handler memilikinya, misalnya dengan mencocokkan agen tersimpan khusus.
4. Jika `environment_connection` masih tertunda, periksa microVM yang direkam dengan`GetMicrovm`. Lanjutkan dengan `ResumeMicrovm` jika ditangguhkan. Panggil `RunMicrovm` hanya jika tidak ada VM yang direkam, lalu simpan ID-nya. Jika penyimpanan gagal, hentikan VM yang Anda luncurkan. Jika pelaksana terhubung sebelum batas waktu koneksi, input menunggu berlangsung tanpa pengiriman ulang.
5. Untuk suatu `agent.session.failed` peristiwa, ambil sesi saat ini dan hentikan microVM yang direkam hanya jika statusnya masih gagal. Abaikan sesi yang dihapus, tindakan yang diselesaikan, dan sesi yang dimiliki oleh handler lain.

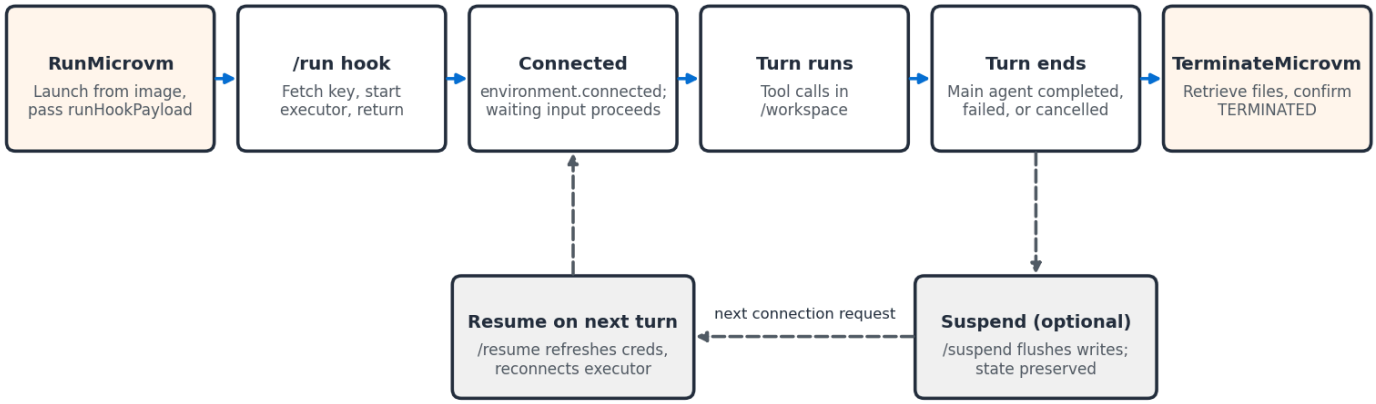
Contoh referensi menyimpan ID microVM dalam metadata sesi. Pengiriman webhook bersamaan dapat meluncurkan VM tambahan sebelum ID itu disimpan. Sebelum penggunaan produksi, cegah peluncuran bersamaan untuk sesi yang sama dan deduplikat peristiwa webhook.

Mengakhiri giliran

Gambar 2 menunjukkan siklus hidup microVM untuk satu putaran, dan jalur penangguhan dan lanjutkan opsional untuk putaran lanjutan.

Figure 2. MicroVM lifecycle for one Agents API turn

Single-turn path (solid) and multi-turn extension with suspend and resume (dashed)



Gambar 2. Siklus hidup MicroVM untuk satu giliran API Agen. `RunMicrovm`meluncurkan microVM, kait `/run` memulai eksekutor, lingkungan terhubung, belokan berjalan, dan belokan berakhir. Di jalur satu putaran, Anda mengambil file dan menelep `TerminateMicrovm` on. Di jalur multi-putaran, microVM menangguhkan setelah belokan dan melanjutkan pada permintaan koneksi berikutnya.

Pada aliran sesi, tunggu`agent.session.turn.completed`,`agent.session.turn.failed`, atau `agent.session.turn.cancelled` untuk agen utama, di `event.turn.subagent_id` mananull. Subagen berbagi microVM, sehingga peristiwa terminal mereka tidak memicu pembersihan. Ambil file yang Anda butuhkan. Setelah putaran terakhir, hubungi `TerminateMicrovm` dan konfirmasikan pencapaian VM`TERMINATED`. Untuk giliran lanjutan, biarkan berjalan atau tunda. Jalankan pembersihan jika aplikasi Anda mengalami kesalahan.

Hasil Turn tiba sebagai acara streaming, bukan langganan webhook. `agent.session.failed`Webhook mencakup kegagalan sesi tetapi tidak setiap giliran yang gagal. Untuk alasan ini, jangan berhenti `agent.session.idle` sendiri, karena dapat tiba sebelum input menunggu dimulai.

Untuk mengikat masa pakai microVM jika pembersihan tidak berjalan, konfigurasikan pengaturan berikut saat peluncuran:

Pengaturan	Bimbingan
`maximumDurationInSeconds`	Tetapkan plafon untuk beban kerja, seperti 900 untuk tes 15 menit. Batas ini berlaku bahkan saat pekerjaan sedang berlangsung.

Pengaturan	Bimbingan
`maxIdleDurationSeconds`	Tutupi beban kerja yang diharapkan. Lambda mengukur lalu lintas masuk, sehingga koneksi keluar pelaksana tidak mengatur ulang timer ini.
`suspendedDurationSeconds` / `autoResumeEnabled`	Gunakan 0/false untuk pola putaran tunggal, yang tidak melanjutkan microVM yang ditangguhkan.

Untuk kontrol seumur hidup dan idle, lihatthe section called “Menjalankan MicroVMS”.

Hapus sesi secara terpisah. Menghapus sesi tidak menghentikan microVM atau mengirim webhook penghapusan. Jaga rahasia gambar dan kunci eksekutor untuk digunakan kembali. Untuk menangani putaran tindak lanjut, lihat Siklus hidup Sandbox.

Menangguhkan dan melanjutkan di antara belokan

Untuk mempertahankan memori dan status disk di antara putaran, tunda microVM setelah giliran agen utama selesai dan semua alat dan subagen selesai. Lanjutkan microVM yang sama ketika sesi berikutnya membutuhkan eksekutor.

- Tetapkan bukan nol`suspendedDurationSeconds`. Biaya penyimpanan snapshot berlaku saat microVM ditangguhkan.
- Aktifkan `/suspend` hook untuk menghapus penulisan dan menutup koneksi, dan `/resume` hook untuk menyegarkan kredenSIAL dan menghubungkan kembali eksekutor.
- Jalur yang dikelola aplikasi melanjutkan VM sebelum mengirim input berikutnya. Di jalur yang dikelola webhook, mengirimkan input memicu permintaan koneksi yang melanjutkan VM. Giliran berlanjut setelah pelaksana terhubung kembali.

> **Note**
>
> `maximumDurationInSeconds`membatasi total waktu berjalan dan ditangguhkan hingga 8 jam. Untuk melanjutkan sesuai permintaan, hubungi`ResumeMicrovm`. Resume otomatis memerlukan lalu lintas masuk ke microVM; Input API agen saja tidak melanjutkannya.

Jaringan

Izinkan koneksi keluar yang diperlukan eksekutor ke OpenAI dan akses ke Secrets Manager. Untuk menjangkau sumber daya pribadi atau menerapkan batasan jaringan Anda sendiri, pasang konektor keluar VPC pada waktu peluncuran. Lihat the section called “Bekerja dengan konektor jaringan jalan keluar”.

Contoh ini mendukung unggahan dan unduhan file melalui titik akhir HTTPS microVM, menggunakan token otentikasi yang dicakup ke port server.

Memantau

Jalankan salah satu contoh, lalu konfirmasikan bahwa `hello.txt` unduhan, microVM mencapai`TERMINATED`, dan sesi yang dihapus mengembalikan HTTP 404.

Log peluncur. Tinjau log fungsi peluncur Lambda di CloudWatch Log:

```
aws logs tail /aws/lambda/codex-agents-api-webhook --since 10m
```

Menjalankan MicroVMS. Set `MICROVM_IMAGE_ARN` el ke gambar ARN Anda, gunakan Wil AWS ayah penerapan Anda, dan daftarkan microVM untuk gambar:

```
aws lambda-microvms list-microvms \
  --image-identifier "$MICROVM_IMAGE_ARN" \
  --query 'items[].{id:microvmId,state:state}' --output table
```

Log sesi dan ID microVM bersama-sama sehingga Anda dapat melacak proses dari ujung ke ujung. Jauhkan kredenSIAL dan badan webhook mentah dari log.

Pemecahan masalah

Gejala	Apa yang harus diperiksa
Tanda tangan Webhook ditolak	Gunakan rahasia penandatanganan titik akhir dan verifikasi badan permintaan yang tidak dimodifikasi.
Tidak ada peluncuran MicroVM	Periksa langganan webhook, filter kepemilikan sesi, `environment_connection` tindakan yang tertunda, dan izin IAM peluncur.

Gejala	Apa yang harus diperiksa
Pembuatan gambar gagal	Periksa artefak Amazon S3, peran pembuatan gambar, dan respons `/ready` kait.
`/run`gagal atau habis waktu	Periksa akses rahasia dan startup pelaksana. Kembali setelah memulai proses, bukan setelah belokan.
Pelaksana tidak dapat terhubung	Periksa ID lingkungan, URL jarak jauh, kepemilikan kunci, dan akses jaringan keluar.
MicroVM berhenti selama belokan	Periksa kebijakan masa pakai dan idle maksimum. Lalu lintas pelaksana keluar tidak dihitung sebagai aktivitas inbound.

Untuk kegagalan koneksi, periksa `agent.session.environment.failed` peristiwa dan log pelaksana. Untuk kontrak eksekutor bersama, lihat kotak Self-hosted pasir.

Sumber daya terkait

- MicroVM
- the section called “Menjalankan MicroVMS”
- the section called “Keamanan”
- Referensi API RunMicrovm
- Kotak pasir yang di-host sendiri OpenAI Agents API
- OpenAI Agents API pada panduan AWS

Keamanan di AWS Lambda

Bab ini mencakup keamanan, tata kelola, dan kepatuhan untuk AWS Lambda. Perlindungan data, kepatuhan, ketahanan, dan panduan keamanan infrastruktur berlaku untuk semua jenis komputasi Lambda. Kontrol akses dan enkripsi memiliki detail spesifik jenis komputasi: untuk Fungsi Web Lambda, lihat,, the section called “Kontrol akses” dan; the section called “Enkripsi saat diam” untuk MicroVMS Lambdathe section called “Izin”, lihat. the section called “Keamanan” Topik yang tersisa dalam Bab ini menjelaskan Fungsi Peristiwa Lambda.

Keamanan cloud di AWS adalah prioritas tertinggi. Sebagai AWS pelanggan, Anda mendapat manfaat dari pusat data dan arsitektur jaringan yang dibangun untuk memenuhi persyaratan organisasi yang paling sensitif terhadap keamanan.

Keamanan adalah tanggung jawab bersama antara Anda AWS dan Anda. Model tanggung jawab bersama menggambarkan hal ini sebagai keamanan dari cloud dan keamanan di cloud:

- Keamanan cloud — AWS bertanggung jawab untuk melindungi infrastruktur yang berjalan Layanan AWS di AWS Cloud. AWS juga memberi Anda layanan yang dapat Anda gunakan dengan aman. Third-party Auditor secara teratur menguji dan memverifikasi efektivitas keamanan kami sebagai bagian dari program AWS kepatuhan. Untuk mempelajari tentang program kepatuhan yang berlaku AWS Lambda, lihat Layanan AWS di Cakupan berdasarkan Program Kepatuhan.
- Keamanan di cloud – Tanggung jawab Anda ditentukan menurut layanan AWS yang Anda gunakan. Anda juga bertanggung jawab atas faktor lain termasuk sensitivitas data Anda, persyaratan perusahaan Anda, serta undang-undang dan peraturan yang berlaku.

Dokumentasi ini membantu Anda memahami cara menerapkan model tanggung jawab bersama saat menggunakan Lambda. Topik berikut menunjukkan kepada Anda cara mengonfigurasi Lambda untuk memenuhi tujuan keamanan dan kepatuhan Anda. Anda juga mempelajari cara menggunakan Layanan AWS yang lain yang membantu Anda memantau dan mengamankan sumber daya Lambda Anda. Topik meliputi validasi kepatuhan, perlindungan data, manajemen identitas dan akses, titik akhir VPC antarmuka, dan ketahanan.

Untuk informasi selengkapnya tentang penerapan prinsip keamanan pada aplikasi Lambda, lihat Keamanan di Tanah Tanpa Server.

Topik

- Perlindungan data di AWS Lambda

- Menggunakan peran terkait layanan untuk Lambda
- Identity and Access Management untuk AWS Lambda
- Buat strategi tata kelola untuk fungsi dan lapisan Lambda
- Validasi kepatuhan untuk AWS Lambda
- Ketahanan di AWS Lambda
- Keamanan infrastruktur di AWS Lambda
- Mengamankan beban kerja dengan titik akhir publik
- Menggunakan penandatanganan kode untuk memverifikasi integritas kode dengan Lambda

Perlindungan data di AWS Lambda

> **Note**
>
> Detail enkripsi dalam topik ini, termasuk enkripsi variabel lingkungan, menjelaskan Fungsi Peristiwa Lambda. Untuk enkripsi Fungsi Web Lambda, lihat. the section called “Enkripsi saat diam”

Model tanggung jawab bersama AWS berlaku untuk perlindungan data di AWS Lambda. Seperti yang dijelaskan dalam model AWS ini, bertanggung jawab untuk melindungi infrastruktur global yang menjalankan semua AWS Cloud. Anda bertanggung jawab untuk mempertahankan kendali atas konten yang di-host pada infrastruktur ini. Anda juga bertanggung jawab atas tugas-tugas konfigurasi dan manajemen keamanan untuk Layanan AWS yang Anda gunakan. Untuk informasi selengkapnya tentang privasi data, lihat FAQ Privasi Data AWS . Untuk informasi tentang perlindungan data di Eropa, lihat Pusat Peraturan Umum Perlindungan Data (GDPR).

Untuk tujuan perlindungan data, kami menyarankan Anda melindungi Akun AWS kredensyal dan mengatur pengguna individu dengan Pusat Identitas AWS IAM atau AWS Identity and Access Management (IAM). Dengan cara itu, setiap pengguna hanya diberi izin yang diperlukan untuk memenuhi tanggung jawab tugasnya. Kami juga menyarankan supaya Anda mengamankan data dengan cara-cara berikut:

- Gunakan autentikasi multi-faktor (MFA) pada setiap akun.
- Gunakan SSL/TLS untuk berkomunikasi dengan AWS sumber daya. Kami mensyaratkan TLS 1.2 dan menganjurkan TLS 1.3.

- Siapkan API dan pencatatan aktivitas pengguna dengan AWS CloudTrail. Untuk informasi tentang penggunaan CloudTrail jejak untuk menangkap AWS aktivitas, lihat Bekerja dengan CloudTrail jejak di AWS CloudTrail Panduan Pengguna.
- Gunakan solusi AWS enkripsi, bersama dengan semua kontrol keamanan default di dalamnya Layanan AWS.
- Gunakan layanan keamanan terkelola tingkat lanjut seperti Amazon Macie, yang membantu menemukan dan mengamankan data sensitif yang disimpan di Amazon S3.
- Jika Anda memerlukan modul kriptografi tervalidasi FIPS 140-3 saat mengakses AWS melalui antarmuka baris perintah atau API, gunakan titik akhir FIPS. Lihat informasi selengkapnya tentang titik akhir FIPS yang tersedia di Standar Pemrosesan Informasi Federal (FIPS) 140-3.

Kami sangat merekomendasikan agar Anda tidak pernah memasukkan informasi identifikasi yang sensitif, seperti nomor rekening pelanggan Anda, ke dalam tanda atau bidang isian bebas seperti bidang Nama. Ini termasuk saat Anda bekerja dengan Lambda atau lainnya Layanan AWS menggunakan konsol, API AWS CLI, atau AWS SDK. Data apa pun yang Anda masukkan ke dalam tanda atau bidang isian bebas yang digunakan untuk nama dapat digunakan untuk log penagihan atau log diagnostik. Saat Anda memberikan URL ke server eksternal, kami sangat menganjurkan supaya Anda tidak menyertakan informasi kredensial di dalam URL untuk memvalidasi permintaan Anda ke server itu.

Bagian-bagian

- Enkripsi dalam transit
- Enkripsi data saat istirahat untuk AWS Lambda

Enkripsi dalam transit

Titik akhir API Lambda hanya mendukung koneksi aman melalui HTTPS. Saat Anda mengelola sumber daya Lambda dengan Konsol Manajemen AWS,AWS SDK, atau Lambda API, semua komunikasi dienkripsi dengan Transport Layer Security (TLS). Untuk daftar lengkap titik akhir API, lihat AWS Wilayah dan titik akhir di. Referensi Umum AWS

Saat Anda menghubungkan fungsi Anda ke sistem file, Lambda menggunakan enkripsi dalam perjalanan untuk semua koneksi. Untuk informasi selengkapnya, lihat Enkripsi data di Amazon EFS di Panduan Pengguna Amazon Elastic File System.

Saat Anda menggunakan variabel lingkungan, Anda dapat mengaktifkan pembantu enkripsi konsol untuk menggunakan enkripsi sisi klien untuk melindungi variabel lingkungan yang sedang transit. Lihat informasi yang lebih lengkap di Mengamankan variabel lingkungan Lambda.

Enkripsi data saat istirahat untuk AWS Lambda

AWS Lambda selalu menyediakan enkripsi saat istirahat untuk sumber daya berikut:

- Variabel-variabel lingkungan
- File yang Anda unggah ke Lambda, termasuk paket penerapan dan arsip lapisan
- objek kriteria filter pemetaan sumber peristiwa
- Data eksekusi yang tahan lama

> **Note**
>
> Untuk fungsi dan lapisan yang menggunakan penyimpanan kode S3 yang dikelola sendiri, Lambda tidak menyimpan salinan kode sumber Anda. Kode Anda tetap berada di bucket S3 Anda, dan enkripsi saat istirahat untuk kode sumber Anda dikelola oleh konfigurasi bucket S3 Anda. Lambda akan digunakan `KMSKeyArn` untuk mengenkripsi versi unzip dari paket yang digunakan Lambda untuk menjalankan fungsi. Untuk informasi selengkapnya, lihat Melindungi data dengan enkripsi sisi server di Panduan Pengguna Layanan Penyimpanan Sederhana Amazon.

Anda dapat mengonfigurasi Lambda secara opsional untuk menggunakan kunci yang dikelola pelanggan untuk mengenkripsi variabel lingkungan, paket penerapan .zip, objek kriteria filter, dan data eksekusi yang tahan lama.

Amazon CloudWatch Logs dan AWS X-Ray juga mengenkripsi data secara default, dan dapat dikonfigurasi untuk menggunakan kunci yang dikelola pelanggan. Untuk detailnya, lihat Mengenkripsi data CloudWatch log di Log dan Perlindungan data di AWS X-Ray.

Memantau kunci enkripsi Anda untuk Lambda

Saat Anda menggunakan kunci yang dikelola AWS KMS pelanggan dengan Lambda, Anda dapat menggunakannya. AWS CloudTrail Contoh berikut adalah CloudTrail acara untuk`Decrypt`,`DescribeKey`, dan `GenerateDataKey` panggilan yang dilakukan oleh Lambda untuk mengakses data yang dienkripsi oleh kunci yang dikelola pelanggan Anda.

Untuk CloudTrail contoh khusus untuk data eksekusi tahan lama, lihatMemantau kunci KMS untuk fungsi yang tahan lama.

Decrypt

Jika Anda menggunakan kunci terkelola AWS KMS pelanggan untuk mengenkripsi objek kriteria filter, Lambda mengirimkan `Decrypt` permintaan atas nama Anda saat Anda mencoba mengaksesnya dalam teks biasa (misalnya, dari panggilan). `ListEventSourceMappings` Contoh peristiwa berikut mencatat `Decrypt` operasi:


```
{
    "eventVersion": "1.09",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROA123456789EXAMPLE:example",
        "arn": "arn:aws:sts::123456789012:assumed-role/role-name/example",
        "accountId": "123456789012",
        "accessKeyId": "ASIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROA123456789EXAMPLE",
                "arn": "arn:aws:iam::123456789012:role/role-name",
                "accountId": "123456789012",
                "userName": "role-name"
            },
            "attributes": {
                "creationDate": "2024-05-30T00:45:23Z",
                "mfaAuthenticated": "false"
            }
        },
        "invokedBy": "lambda.amazonaws.com"
    },
    "eventTime": "2024-05-30T01:05:46Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "Decrypt",
    "awsRegion": "eu-west-1",
    "sourceIPAddress": "lambda.amazonaws.com",
    "userAgent": "lambda.amazonaws.com",
    "requestParameters": {
        "keyId": "arn:aws:kms:eu-west-1:123456789012:key/a1b2c3d4-5678-90ab-cdef-
EXAMPLE11111",
        "encryptionContext": {
```

```
            "aws-crypto-public-key": "ABCD
+7876787678+CDEFGHIJKL/888666888999888555444111555222888333111==",
            "aws:lambda:EventSourceArn": "arn:aws:sqs:eu-west-1:123456789012:sample-
source",
            "aws:lambda:FunctionArn": "arn:aws:lambda:eu-
west-1:123456789012:function:sample-function"
        },
        "encryptionAlgorithm": "SYMMETRIC_DEFAULT"
    },
    "responseElements": null,
    "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEaaaaa",
    "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbb",
    "readOnly": true,
    "resources": [
        {
            "accountId": "AWS Internal",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:eu-west-1:123456789012:key/a1b2c3d4-5678-90ab-cdef-
EXAMPLE11111"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "123456789012",
    "eventCategory": "Management",
    "sessionCredentialFromConsole": "true"
}
```

DescribeKey

Jika Anda menggunakan kunci terkelola AWS KMS pelanggan untuk mengenkripsi objek kriteria filter, Lambda mengirimkan `DescribeKey` permintaan atas nama Anda saat Anda mencoba mengaksesnya (misalnya, dari `GetEventSourceMapping` panggilan). Contoh peristiwa berikut mencatat `DescribeKey` operasi:

```
{
    "eventVersion": "1.09",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROA123456789EXAMPLE:example",
        "arn": "arn:aws:sts::123456789012:assumed-role/role-name/example",
        "accountId": "123456789012",
        "accessKeyId": "ASIAIOSFODNN7EXAMPLE",
```

```
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROA123456789EXAMPLE",
                "arn": "arn:aws:iam::123456789012:role/role-name",
                "accountId": "123456789012",
                "userName": "role-name"
            },
            "attributes": {
                "creationDate": "2024-05-30T00:45:23Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2024-05-30T01:09:40Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "DescribeKey",
    "awsRegion": "eu-west-1",
    "sourceIPAddress": "54.240.197.238",
    "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36
 (KHTML, like Gecko) Chrome/125.0.0.0 Safari/537.36",
    "requestParameters": {
        "keyId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
    },
    "responseElements": null,
    "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEaaaaa",
    "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbb",
    "readOnly": true,
    "resources": [
        {
            "accountId": "AWS Internal",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:eu-west-1:123456789012:key/a1b2c3d4-5678-90ab-cdef-
EXAMPLE11111"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "123456789012",
    "eventCategory": "Management",
    "tlsDetails": {
        "tlsVersion": "TLSv1.3",
        "cipherSuite": "TLS_AES_256_GCM_SHA384",
        "clientProvidedHostHeader": "kms.eu-west-1.amazonaws.com"
```

```
    },
    "sessionCredentialFromConsole": "true"
}
```

GenerateDataKey

Saat Anda menggunakan kunci yang dikelola AWS KMS pelanggan untuk mengenkripsi objek kriteria filter Anda dalam `UpdateEventSourceMapping` panggilan `CreateEventSourceMapping` atau, Lambda mengirimkan `GenerateDataKey` permintaan atas nama Anda untuk menghasilkan kunci data untuk mengenkripsi kriteria filter (enkripsi amplop). Contoh peristiwa berikut mencatat `GenerateDataKey` operasi:

```
{
    "eventVersion": "1.09",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROA123456789EXAMPLE:example",
        "arn": "arn:aws:sts::123456789012:assumed-role/role-name/example",
        "accountId": "123456789012",
        "accessKeyId": "ASIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROA123456789EXAMPLE",
                "arn": "arn:aws:iam::123456789012:role/role-name",
                "accountId": "123456789012",
                "userName": "role-name"
            },
            "attributes": {
                "creationDate": "2024-05-30T00:06:07Z",
                "mfaAuthenticated": "false"
            }
        },
        "invokedBy": "lambda.amazonaws.com"
    },
    "eventTime": "2024-05-30T01:04:18Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "GenerateDataKey",
    "awsRegion": "eu-west-1",
    "sourceIPAddress": "lambda.amazonaws.com",
    "userAgent": "lambda.amazonaws.com",
    "requestParameters": {
        "numberOfBytes": 32,
```

```
        "keyId": "arn:aws:kms:eu-west-1:123456789012:key/a1b2c3d4-5678-90ab-cdef-
EXAMPLE11111",
        "encryptionContext": {
            "aws-crypto-public-key": "ABCD
+7876787678+CDEFGHIJKL/888666888999888555444111555222888333111==",
            "aws:lambda:EventSourceArn": "arn:aws:sqs:eu-west-1:123456789012:sample-
source",
            "aws:lambda:FunctionArn": "arn:aws:lambda:eu-
west-1:123456789012:function:sample-function"
        },
    },
    "responseElements": null,
    "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEaaaaa",
    "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbb",
    "readOnly": true,
    "resources": [
        {
            "accountId": "AWS Internal",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:eu-west-1:123456789012:key/a1b2c3d4-5678-90ab-cdef-
EXAMPLE11111"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "123456789012",
    "eventCategory": "Management"
}
```


Menggunakan peran terkait layanan untuk Lambda

Lambda menggunakan peran AWS Identity and Access Management terkait layanan (IAM). Peran terkait layanan adalah jenis peran IAM unik yang ditautkan langsung ke Lambda. Service-linked peran telah ditentukan sebelumnya oleh Lambda dan menyertakan izin yang diperlukan layanan untuk memanggil AWS layanan lain atas nama Anda.

Lambda mendefinisikan izin peran terkait layanan, dan hanya Lambda yang dapat mengambil perannya. Izin yang ditentukan mencakup kebijakan kepercayaan dan kebijakan izin, dan kebijakan izin tersebut tidak dapat dilampirkan ke entitas IAM lainnya.

Anda dapat menghapus peran tertaut layanan hanya setelah menghapus sumber daya terkait terlebih dahulu. Ini melindungi sumber daya Lambda Anda karena Anda tidak dapat menghapus izin untuk mengakses sumber daya secara tidak sengaja.

Untuk informasi tentang layanan lain yang mendukung peran terkait layanan, lihat AWS layanan yang bekerja dengan IAM dan cari layanan yang memiliki Ya di kolom peran. Service-linked Pilih Ya dengan sebuah tautan untuk melihat dokumentasi peran terkait layanan untuk layanan tersebut.

Service-linked izin peran untuk Lambda

Lambda menggunakan peran terkait layanan bernama. AWSServiceRoleForLambda Peran tertaut layanan memercayai layanan berikut untuk mengambil peran tersebut:

- `lambda.amazonaws.com`

Kebijakan izin peran yang dinamai AWSLambdaServiceRolePolicy memungkinkan Lambda menyelesaikan tindakan berikut pada sumber daya yang ditentukan:

- Tind `ec2:TerminateInstances` akan: `arn:aws:ec2:*:*:instance/*` aktif dengan kondisi yang `ec2:ManagedResourceOperator` sama `scaler.lambda.amazonaws.com`
- Aksi: `ec2:DescribeInstanceStatus` dan seter `ec2:DescribeInstances` usnya `*`

Anda harus mengonfigurasi izin untuk mengizinkan pengguna, grup, atau peran Anda membuat, mengedit, atau menghapus peran terkait layanan. Untuk informasi selengkapnya, lihat izin Service-linked peran di Panduan Pengguna IAM.

Untuk pembaruan kebijakan terkelola, lihat Kebijakan terkelola Lambda.

Membuat peran terkait layanan untuk Lambda

Anda tidak perlu membuat peran terkait layanan secara manual. Saat Anda membuat penyedia kapasitas Lambda di Konsol Manajemen AWS,, atau AWS API AWS CLI, Lambda membuat peran terkait layanan untuk Anda.

Jika Anda menghapus peran terkait layanan ini, dan ingin membuatnya lagi, Anda dapat mengulangi proses yang sama untuk membuat kembali peran tersebut di akun Anda. Saat Anda membuat penyedia kapasitas Lambda, Lambda membuat peran terkait layanan untuk Anda lagi.

Anda juga dapat menggunakan konsol IAM untuk membuat peran terkait layanan dengan kasus penggunaan. AWSServiceRoleForLambda Di AWS API AWS CLI atau, buat peran terkait layanan dengan nama `lambda.amazonaws.com` layanan. Untuk informasi selengkapnya, lihat Membuat peran tertaut layanan dalam Panduan Pengguna IAM. Jika Anda menghapus peran tertaut layanan ini, Anda dapat mengulang proses yang sama untuk membuat peran tersebut lagi.

Mengedit peran terkait layanan untuk Lambda

Lambda tidak mengizinkan Anda mengedit peran terkait AWSServiceRoleForLambda layanan. Setelah membuat peran terkait layanan, Anda tidak dapat mengubah nama peran karena berbagai entitas mungkin merujuk peran tersebut. Namun, Anda dapat mengedit penjelasan peran menggunakan IAM. Untuk informasi selengkapnya, lihat Mengedit peran terkait layanan dalam Panduan Pengguna IAM.

Menghapus peran terkait layanan untuk Lambda

Jika Anda tidak perlu lagi menggunakan fitur atau layanan yang memerlukan peran terkait layanan, kami merekomendasikan Anda menghapus peran tersebut. Dengan begitu, Anda tidak memiliki entitas yang tidak digunakan yang tidak dipantau atau dipelihara secara aktif. Tetapi, Anda harus membersihkan sumber daya peran terkait layanan sebelum menghapusnya secara manual.

> **Note**
>
> Jika layanan Lambda menggunakan peran saat Anda mencoba menghapus sumber daya, maka penghapusan mungkin gagal. Jika hal itu terjadi, tunggu beberapa menit dan coba mengoperasikannya lagi.

Untuk menghapus sumber daya Lambda yang digunakan oleh AWSServiceRoleForLambda

1. Hapus semua penyedia kapasitas Lambda dari akun Anda. Anda dapat melakukan ini menggunakan konsol Lambda, CLI, atau API.
2. Pastikan tidak ada penyedia kapasitas Lambda yang tersisa di akun Anda sebelum mencoba menghapus peran terkait layanan.

Untuk menghapus peran tertaut layanan secara manual menggunakan IAM

Gunakan konsol IAM AWS CLI, atau AWS API untuk menghapus peran terkait AWSServiceRoleForLambda layanan. Untuk informasi selengkapnya, lihat Menghapus peran terkait layanan dalam Panduan Pengguna IAM.

Wilayah yang Didukung untuk peran terkait layanan Lambda

Lambda tidak mendukung penggunaan peran terkait layanan di setiap Wilayah tempat layanan tersedia. AWSServiceRoleForLambda didukung di Wilayah berikut.

Nama wilayah	Identitas wilayah	Dukungan dalam Lambda
US East (Northern Virginia)	us-east-1	Ya
US East (Ohio)	us-east-2	Ya
US West (N. California)	us-west-1	Ya
US West (Oregon)	us-west-2	Ya
Africa (Cape Town)	af-south-1	Tidak
Asia Pasifik (Hong Kong)	ap-east-1	Ya
Asia Pasifik (Jakarta)	ap-southeast-3	Ya
Asia Pasifik (Bangkok)	ap-Southeast-7	Ya
Asia Pacific (Mumbai)	ap-south-1	Ya
Asia Pacific (Osaka)	ap-northeast-3	Tidak
Asia Pasifik (Seoul)	ap-northeast-2	Tidak
Asia Pasifik (Singapura)	ap-southeast-1	Ya
Asia Pacific (Sydney)	ap-southeast-2	Ya
Asia Pacific (Tokyo)	ap-northeast-1	Ya
Canada (Central)	ca-central-1	Tidak

Nama wilayah	Identitas wilayah	Dukungan dalam Lambda
Eropa (Frankfurt)	eu-central-1	Ya
Eropa (Irlandia)	eu-west-1	Ya
Eropa (London)	eu-west-2	Ya
Europe (Milan)	eu-south-1	Tidak
Eropa (Paris)	eu-west-3	Tidak
Eropa (Stockholm)	eu-north-1	Tidak
Timur Tengah (Bahrain)	me-south-1	Tidak
Timur Tengah (UAE)	me-central-1	Tidak
Amerika Selatan (Sao Paulo)	sa-east-1	Tidak
AWS GovCloud (US-East)	us-gov-east-1	Tidak
AWS GovCloud (US-West)	us-gov-west-1	Tidak

Identity and Access Management untuk AWS Lambda

> **Note**
>
> Panduan kebijakan berbasis identitas dan sumber daya dalam topik ini menjelaskan Fungsi Peristiwa Lambda. Fungsi Web Lambda menggunakan model kontrol akses yang berbeda; lihat dan. the section called “Izin” the section called “Kontrol akses” Untuk MicroVMS Lambda, lihat. the section called “Keamanan”

AWS Identity and Access Management (IAM) adalah Layanan AWS yang membantu administrator mengontrol akses ke AWS sumber daya dengan aman. Administrator IAM mengontrol siapa yang dapat diautentikasi (masuk) dan diotorisasi (memiliki izin) untuk menggunakan sumber daya Lambda. IAM adalah Layanan AWS yang dapat Anda gunakan tanpa biaya tambahan.

Topik

- Audiens
- Mengautentikasi dengan identitas
- Mengelola akses menggunakan kebijakan
- Bagaimana AWS Lambda bekerja dengan IAM
- Identity-based contoh kebijakan untuk AWS Lambda
- AWS kebijakan terkelola untuk AWS Lambda
- Pemecahan masalah AWS Lambda identitas dan akses

Audiens

Cara Anda menggunakan AWS Identity and Access Management (IAM) berbeda berdasarkan peran Anda:

- Pengguna layanan - minta izin dari administrator Anda jika Anda tidak dapat mengakses fitur (lihat Pemecahan masalah AWS Lambda identitas dan akses)
- Administrator layanan - tentukan akses pengguna dan mengirimkan permintaan izin (lihat Bagaimana AWS Lambda bekerja dengan IAM)
- Administrator IAM - tulis kebijakan untuk mengelola akses (lihat Identity-based contoh kebijakan untuk AWS Lambda)

Mengautentikasi dengan identitas

Otentikasi adalah cara Anda masuk AWS menggunakan kredensi identitas Anda. Anda harus diautentikasi sebagai Pengguna root akun AWS, pengguna IAM, atau dengan mengasumsikan peran IAM.

Anda dapat masuk sebagai identitas federasi menggunakan kredensil dari sumber identitas seperti Pusat Identitas AWS IAM (Pusat Identitas IAM), autentikasi masuk tunggal, atau kredensional. Google/Facebook Untuk informasi selengkapnya tentang cara masuk, lihat Cara masuk ke Akun AWS Anda dalam Panduan Pengguna AWS Sign-In .

Untuk akses terprogram, AWS sediakan SDK dan CLI untuk menandatangani permintaan secara kriptografis. Untuk informasi selengkapnya, lihat AWS Signature Version 4 untuk permintaan API dalam Panduan Pengguna IAM.

Akun AWS pengguna root

Saat Anda membuat Akun AWS, Anda mulai dengan satu identitas masuk yang disebut pengguna Akun AWS root yang memiliki akses lengkap ke semua Layanan AWS dan sumber daya. Kami sangat menyarankan agar Anda tidak menggunakan pengguna root untuk tugas sehari-hari. Untuk tugas yang memerlukan kredensial pengguna root, lihat Tugas yang memerlukan kredensial pengguna root dalam Panduan Pengguna IAM.

Identitas terfederasi

Sebagai praktik terbaik, mewajibkan pengguna manusia untuk menggunakan federasi dengan penyedia identitas untuk mengakses Layanan AWS menggunakan kredensi sementara.

Identitas federasi adalah pengguna dari direktori perusahaan Anda, penyedia identitas web, atau Directory Service yang mengakses Layanan AWS menggunakan kredensil dari sumber identitas. Identitas terfederasi mengambil peran yang memberikan kredensial sementara.

Untuk manajemen akses terpusat, kami menyarankan Pusat Identitas AWS IAM. Untuk informasi selengkapnya, lihat Apa itu Pusat Identitas IAM? dalam Panduan Pengguna Pusat Identitas AWS IAM .

Pengguna dan grup IAM

Pengguna IAM adalah identitas dengan izin khusus untuk satu orang atau aplikasi. Sebaiknya gunakan kredensial sementara alih-alih pengguna IAM dengan kredensial jangka panjang. Untuk informasi selengkapnya, lihat Mewajibkan pengguna manusia untuk menggunakan federasi dengan penyedia identitas untuk mengakses AWS menggunakan kredensi sementara di Panduan Pengguna IAM.

Grup IAM menentukan kumpulan pengguna IAM dan mempermudah pengelolaan izin untuk pengguna dalam jumlah besar. Untuk mempelajari selengkapnya, lihat Kasus penggunaan untuk pengguna IAM dalam Panduan Pengguna IAM.

Peran IAM

Peran IAM adalah identitas dengan izin khusus yang menyediakan kredensial sementara. Anda dapat mengambil peran dengan beralih dari pengguna ke peran IAM (konsol) atau dengan memanggil operasi AWS CLI atau AWS API. Untuk informasi selengkapnya, lihat Metode untuk mengambil peran dalam Panduan Pengguna IAM.

Peran IAM berguna untuk akses pengguna terfederasi, izin pengguna IAM sementara, akses lintas akun, akses lintas layanan, dan aplikasi yang berjalan di Amazon EC2. Untuk informasi selengkapnya, lihat Akses sumber daya lintas akun di IAM dalam Panduan Pengguna IAM.

Mengelola akses menggunakan kebijakan

Anda mengontrol akses AWS dengan membuat kebijakan dan melampirkannya ke AWS identitas atau sumber daya. Kebijakan menentukan izin saat dikaitkan dengan identitas atau sumber daya. AWS mengevaluasi kebijakan ini ketika kepala sekolah membuat permintaan. Sebagian besar kebijakan disimpan AWS sebagai dokumen JSON. Untuk informasi selengkapnya tentang dokumen kebijakan JSON, lihat Gambaran umum kebijakan JSON dalam Panduan Pengguna IAM.

Menggunakan kebijakan, administrator menentukan siapa yang memiliki akses ke apa dengan mendefinisikan principal mana yang dapat melakukan tindakan pada sumber daya apa, dan dalam kondisi apa.

Secara default, pengguna dan peran tidak memiliki izin. Administrator IAM membuat kebijakan IAM dan menambahkannya ke peran, yang kemudian dapat diambil oleh pengguna. Kebijakan IAM mendefinisikan izin terlepas dari metode yang Anda gunakan untuk melakukan operasinya.

Identity-based kebijakan

Identity-based kebijakan adalah dokumen kebijakan izin JSON yang Anda lampirkan ke identitas (pengguna, grup, atau peran). Kebijakan ini mengontrol tindakan apa yang bisa dilakukan oleh identitas tersebut, terhadap sumber daya yang mana, dan dalam kondisi apa. Untuk mempelajari cara membuat kebijakan berbasis identitas, lihat Tentukan izin IAM kustom dengan kebijakan yang dikelola pelanggan dalam Panduan Pengguna IAM.

Identity-based kebijakan dapat berupa kebijakan inline (disematkan langsung ke dalam satu identitas) atau kebijakan terkelola (kebijakan mandiri yang dilampirkan pada beberapa identitas). Untuk mempelajari cara memilih antara kebijakan terkelola dan kebijakan inline, lihat Pilih antara kebijakan terkelola dan kebijakan inline dalam Panduan Pengguna IAM.

Resource-based kebijakan

Resource-based kebijakan adalah dokumen kebijakan JSON yang Anda lampirkan ke sumber daya. Contohnya termasuk kebijakan kepercayaan peran IAM dan kebijakan bucket Amazon S3. Dalam layanan yang mendukung kebijakan berbasis sumber daya, administrator layanan dapat menggunakannya untuk mengontrol akses ke sumber daya tertentu. Anda harus menentukan principal dalam kebijakan berbasis sumber daya.

Resource-based kebijakan adalah kebijakan inline yang terletak di layanan tersebut. Anda tidak dapat menggunakan kebijakan AWS terkelola dari IAM dalam kebijakan berbasis sumber daya.

Jenis-jenis kebijakan lain

AWS mendukung jenis kebijakan tambahan yang dapat menetapkan izin maksimum yang diberikan oleh jenis kebijakan yang lebih umum:

- Batasan izin – Menetapkan izin maksimum yang dapat diberikan oleh kebijakan berbasis identitas kepada entitas IAM. Untuk informasi selengkapnya, lihat Batasan izin untuk entitas IAM dalam Panduan Pengguna IAM.
- Kebijakan kontrol layanan (SCP) – Menentukan izin maksimum untuk organisasi atau unit organisasi di AWS Organizations. Untuk informasi selengkapnya, lihat Kebijakan kontrol layanan dalam Panduan Pengguna AWS Organizations .
- Kebijakan kontrol sumber daya (RCP) – Menetapkan izin maksimum yang tersedia untuk sumber daya di akun Anda. Untuk informasi selengkapnya, lihat Kebijakan kontrol sumber daya (RCP) dalam Panduan Pengguna AWS Organizations .
- Kebijakan sesi – Kebijakan lanjutan yang diteruskan sebagai parameter saat membuat sesi sementara untuk peran atau pengguna terfederasi. Untuk informasi selengkapnya, lihat Kebijakan sesi dalam Panduan Pengguna IAM.

Berbagai jenis kebijakan

Ketika beberapa jenis kebijakan berlaku pada suatu permintaan, izin yang dihasilkan lebih rumit untuk dipahami. Untuk mempelajari cara AWS menentukan apakah akan mengizinkan permintaan saat beberapa jenis kebijakan terlibat, lihat Logika evaluasi kebijakan di Panduan Pengguna IAM.

Bagaimana AWS Lambda bekerja dengan IAM

Sebelum Anda menggunakan IAM untuk mengelola akses ke Lambda, pelajari fitur IAM apa yang tersedia untuk digunakan dengan Lambda.

Fitur IAM	Dukungan Lambda
Identity-based kebijakan	Ya
Resource-based kebijakan	Ya

Fitur IAM	Dukungan Lambda
Tindakan kebijakan	Ya
Sumber daya kebijakan	Ya
kunci-kunci persyaratan kebijakan (spesifik layanan)	Ya
ACL	Tidak
ABAC (tanda dalam kebijakan)	Parsial
Kredensial sementara	Ya
Sesi akses teruskan (FAS)	Tidak
Peran layanan	Ya
Service-linked peran	Parsial

Untuk mendapatkan tampilan tingkat tinggi tentang cara kerja Lambda dan layanan AWS lainnya dengan sebagian besar fitur IAM, AWS lihat layanan yang bekerja dengan IAM di Panduan Pengguna IAM.

Identity-based kebijakan untuk Lambda

Mendukung kebijakan berbasis identitas: Ya

Identity-based kebijakan adalah dokumen kebijakan izin JSON yang dapat Anda lampirkan ke identitas, seperti pengguna IAM, grup pengguna, atau peran. Kebijakan ini mengontrol jenis tindakan yang dapat dilakukan oleh pengguna dan peran, di sumber daya mana, dan berdasarkan kondisi seperti apa. Untuk mempelajari cara membuat kebijakan berbasis identitas, lihat Tentukan izin IAM kustom dengan kebijakan terkelola pelanggan dalam Panduan Pengguna IAM.

Dengan kebijakan berbasis identitas IAM, Anda dapat menentukan secara spesifik apakah tindakan dan sumber daya diizinkan atau ditolak, serta kondisi yang menjadi dasar dikabulkan atau ditolaknya tindakan tersebut. Untuk mempelajari semua elemen yang dapat Anda gunakan dalam kebijakan JSON, lihat Referensi elemen kebijakan JSON IAM dalam Panduan Pengguna IAM.

Identity-based contoh kebijakan untuk Lambda

Untuk melihat contoh kebijakan berbasis identitas Lambda, lihat. Identity-based contoh kebijakan untuk AWS Lambda

Resource-based Kebijakan dalam Lambda

Mendukung kebijakan berbasis sumber daya: Ya

Resource-based kebijakan adalah dokumen kebijakan JSON yang Anda lampirkan ke sumber daya. Contoh kebijakan berbasis sumber daya adalah kebijakan kepercayaan peran IAM dan kebijakan bucket Amazon S3. Dalam layanan yang mendukung kebijakan berbasis sumber daya, administrator layanan dapat menggunakannya untuk mengontrol akses ke sumber daya tertentu. Untuk sumber daya tempat kebijakan dilampirkan, kebijakan menentukan tindakan apa yang dapat dilakukan oleh principal tertentu pada sumber daya tersebut dan dalam kondisi apa. Anda harus menentukan principal dalam kebijakan berbasis sumber daya. Prinsipal dapat mencakup akun, pengguna, peran, pengguna federasi, atau. Layanan AWS

Untuk mengaktifkan akses lintas akun, Anda dapat menentukan secara spesifik seluruh akun atau entitas IAM di akun lain sebagai principal dalam kebijakan berbasis sumber daya. Untuk informasi selengkapnya, lihat Akses sumber daya lintas akun di IAM dalam Panduan Pengguna IAM.

Anda dapat melampirkan kebijakan berbasis sumber daya ke fungsi atau lapisan Lambda. Kebijakan ini mendefinisikan prinsipal mana yang dapat melakukan tindakan pada fungsi atau lapisan.

Untuk mempelajari cara melampirkan kebijakan berbasis sumber daya ke fungsi atau lapisan, lihat. Bekerja dengan kebijakan berbasis sumber daya di Lambda

Tindakan kebijakan untuk Lambda

Mendukung tindakan kebijakan: Ya

Administrator dapat menggunakan kebijakan AWS JSON untuk menentukan siapa yang memiliki akses ke apa. Yaitu, di mana utama dapat melakukan tindakan pada sumber daya, dan dalam kondisi apa.

Elemen `Action` dari kebijakan JSON menjelaskan tindakan yang dapat Anda gunakan untuk mengizinkan atau menolak akses dalam sebuah kebijakan. Sertakan tindakan dalam kebijakan untuk memberikan izin untuk melakukan operasi terkait.

Untuk melihat daftar tindakan Lambda, lihat Tindakan yang ditentukan oleh AWS Lambda dalam Referensi Otorisasi Layanan.

Tindakan kebijakan di Lambda menggunakan awalan berikut sebelum tindakan:

```
lambda
```

Untuk menetapkan secara spesifik beberapa tindakan dalam satu pernyataan, pisahkan tindakan tersebut dengan koma.

```
"Action": [
      "lambda:action1",
      "lambda:action2"
         ]
```

Untuk melihat contoh kebijakan berbasis identitas Lambda, lihat. Identity-based contoh kebijakan untuk AWS Lambda

Sumber daya kebijakan untuk Lambda

Mendukung sumber daya kebijakan: Ya

Administrator dapat menggunakan kebijakan AWS JSON untuk menentukan siapa yang memiliki akses ke apa. Yaitu, di mana utama dapat melakukan tindakan pada sumber daya, dan dalam kondisi apa.

Elemen kebijakan JSON `Resource` menentukan objek yang menjadi target penerapan tindakan. Praktik terbaiknya, tentukan sumber daya menggunakan Amazon Resource Name (ARN). Untuk tindakan yang tidak mendukung izin di tingkat sumber daya, gunakan wildcard (*) untuk menunjukkan bahwa pernyataan tersebut berlaku untuk semua sumber daya.

```
"Resource": "*"
```

Untuk melihat daftar jenis sumber daya Lambda dan ARNnya, lihat Jenis sumber daya yang ditentukan oleh AWS Lambda dalam Referensi Otorisasi Layanan. Untuk mempelajari dengan tindakan mana Anda dapat menentukan ARN setiap sumber daya, lihat Tindakan yang ditentukan oleh AWS Lambda.

Untuk melihat contoh kebijakan berbasis identitas Lambda, lihat. Identity-based contoh kebijakan untuk AWS Lambda

Kunci kondisi kebijakan untuk Lambda

Mendukung kunci kondisi kebijakan khusus layanan: Yes

Administrator dapat menggunakan kebijakan AWS JSON untuk menentukan siapa yang memiliki akses ke apa. Yaitu, principal dapat melakukan tindakan pada suatu sumber daya, dan dalam suatu syarat.

Elemen `Condition` menentukan ketika pernyataan dieksekusi berdasarkan kriteria yang ditetapkan. Anda dapat membuat ekspresi bersyarat yang menggunakan operator kondisi, misalnya sama dengan atau kurang dari, untuk mencocokkan kondisi dalam kebijakan dengan nilai-nilai yang diminta. Untuk melihat semua kunci kondisi AWS global, lihat kunci konteks kondisi AWS global di Panduan Pengguna IAM.

Untuk melihat daftar kunci kondisi Lambda, lihat Kunci kondisi untuk AWS Lambda Referensi Otorisasi Layanan. Untuk mempelajari tindakan dan sumber daya yang dapat Anda gunakan kunci kondisi, lihat Tindakan yang ditentukan oleh AWS Lambda.

Untuk melihat contoh kebijakan berbasis identitas Lambda, lihat. Identity-based contoh kebijakan untuk AWS Lambda

ACL di Lambda

Mendukung ACL: Tidak

Daftar kontrol akses (ACL) mengendalikan principal mana (anggota akun, pengguna, atau peran) yang memiliki izin untuk mengakses sumber daya. ACL serupa dengan kebijakan berbasis sumber daya, meskipun kebijakan tersebut tidak menggunakan format dokumen kebijakan JSON.

ABAC dengan Lambda

Mendukung ABAC (tag dalam kebijakan): Sebagian

Attribute-based Access Control (ABAC) adalah strategi otorisasi yang mendefinisikan izin berdasarkan atribut yang disebut tag. Anda dapat melampirkan tag ke entitas dan AWS sumber daya IAM, lalu merancang kebijakan ABAC untuk mengizinkan operasi saat tag prinsipal cocok dengan tag pada sumber daya.

Untuk mengendalikan akses berdasarkan tanda, berikan informasi tentang tanda di elemen kondisi dari kebijakan menggunakan kunci kondisi `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, atau `aws:TagKeys`.

Jika sebuah layanan mendukung ketiga kunci kondisi untuk setiap jenis sumber daya, nilainya adalah Ya untuk layanan tersebut. Jika suatu layanan mendukung ketiga kunci kondisi untuk hanya beberapa jenis sumber daya, nilainya adalah Parsial.

Untuk informasi selengkapnya tentang ABAC, lihat Tentukan izin dengan otorisasi ABAC dalam Panduan Pengguna IAM. Untuk melihat tutorial yang menguraikan langkah-langkah pengaturan ABAC, lihat Menggunakan kontrol akses berbasis atribut (ABAC) dalam Panduan Pengguna IAM.

Untuk informasi selengkapnya tentang menandai sumber daya Lambda, lihat. Menggunakan kontrol akses berbasis atribut di Lambda

Menggunakan kredensyal sementara dengan Lambda

Mendukung kredensial sementara: Ya

Kredensi sementara menyediakan akses jangka pendek ke AWS sumber daya dan secara otomatis dibuat saat Anda menggunakan federasi atau beralih peran. AWS merekomendasikan agar Anda secara dinamis menghasilkan kredensi sementara alih-alih menggunakan kunci akses jangka panjang. Untuk informasi selengkapnya, lihat Kredensial keamanan sementara di IAM dan Layanan AWS yang berfungsi dengan IAM dalam Panduan Pengguna IAM.

Teruskan sesi akses untuk Lambda

Mendukung sesi akses maju (FAS): Tidak

Sesi akses teruskan (FAS) menggunakan izin dari pemanggilan utama Layanan AWS, dikombinasikan dengan permintaan Layanan AWS untuk membuat permintaan ke layanan hilir. Untuk detail kebijakan ketika mengajukan permintaan FAS, lihat Sesi akses terusan.

Peran layanan untuk Lambda

Mendukung peran layanan: Ya

Peran layanan adalah peran IAM yang diambil oleh sebuah layanan untuk melakukan tindakan atas nama Anda. Administrator IAM dapat membuat, mengubah, dan menghapus peran layanan dari dalam IAM. Untuk informasi selengkapnya, lihat Buat sebuah peran untuk mendelegasikan izin ke Layanan AWS dalam Panduan pengguna IAM.

Di Lambda, peran layanan dikenal sebagai peran eksekusi.

> **Warning**
>
> Mengubah izin untuk peran eksekusi dapat merusak fungsionalitas Lambda.

Service-linked peran untuk Lambda

Mendukung peran terkait layanan: Sebagian

Peran terkait layanan adalah jenis peran layanan yang ditautkan ke. Layanan AWS Layanan dapat mengambil peran untuk melakukan tindakan atas nama Anda. Service-linked peran muncul di Anda Akun AWS dan dimiliki oleh layanan. Administrator IAM dapat melihat, tetapi tidak dapat mengedit izin untuk peran terkait layanan.

Lambda tidak memiliki peran terkait layanan, tetapi Lambda @Edge memilikinya. Untuk informasi selengkapnya, lihat Service-Linked Peran untuk Lambda @Edge di Panduan CloudFront Pengembang Amazon.

Untuk detail tentang pembuatan atau manajemen peran terkait layanan, lihat Layanan AWS yang berfungsi dengan IAM. Temukan layanan dalam tabel yang `Yes` menyertakan kolom Service-linked peran. Pilih tautan Ya untuk melihat dokumentasi peran terkait layanan untuk layanan tersebut.

Identity-based contoh kebijakan untuk AWS Lambda

Secara default, pengguna dan peran tidak memiliki izin untuk membuat atau memodifikasi sumber daya Lambda. Untuk memberikan izin kepada pengguna untuk melakukan tindakan di sumber daya yang mereka perlukan, administrator IAM dapat membuat kebijakan IAM.

Untuk mempelajari cara membuat kebijakan berbasis identitas IAM dengan menggunakan contoh dokumen kebijakan JSON ini, lihat Membuat kebijakan IAM (konsol) di Panduan Pengguna IAM.

Untuk detail tentang tindakan dan jenis sumber daya yang ditentukan oleh Lambda, termasuk format ARN untuk setiap jenis sumber daya, lihat Kunci tindakan, sumber daya, dan kondisi AWS Lambda di Referensi Otorisasi Layanan.

Topik

- Praktik terbaik kebijakan
- Menggunakan konsol Lambda

- Perbolehkan pengguna untuk melihat izin mereka sendiri

Praktik terbaik kebijakan

Identity-based kebijakan menentukan apakah seseorang dapat membuat, mengakses, atau menghapus sumber daya Lambda di akun Anda. Tindakan ini membuat Akun AWS Anda dikenai biaya. Ketika Anda membuat atau mengedit kebijakan berbasis identitas, ikuti panduan dan rekomendasi ini:

- Mulailah dengan kebijakan AWS terkelola dan beralih ke izin hak istimewa paling sedikit — Untuk mulai memberikan izin kepada pengguna dan beban kerja Anda, gunakan kebijakan AWS terkelola yang memberikan izin untuk banyak kasus penggunaan umum. Mereka tersedia di Anda Akun AWS. Kami menyarankan Anda mengurangi izin lebih lanjut dengan menentukan kebijakan yang dikelola AWS pelanggan yang khusus untuk kasus penggunaan Anda. Untuk informasi selengkapnya, lihat Kebijakan yang dikelola AWS atau Kebijakan yang dikelola AWS untuk fungsi tugas dalam Panduan Pengguna IAM.
- Menerapkan izin dengan hak akses paling rendah – Ketika Anda menetapkan izin dengan kebijakan IAM, hanya berikan izin yang diperlukan untuk melakukan tugas. Anda melakukannya dengan mendefinisikan tindakan yang dapat diambil pada sumber daya tertentu dalam kondisi tertentu, yang juga dikenal sebagai izin dengan hak akses paling rendah. Untuk informasi selengkapnya tentang cara menggunakan IAM untuk mengajukan izin, lihat Kebijakan dan izin dalam IAM dalam Panduan Pengguna IAM.
- Gunakan kondisi dalam kebijakan IAM untuk membatasi akses lebih lanjut – Anda dapat menambahkan suatu kondisi ke kebijakan Anda untuk membatasi akses ke tindakan dan sumber daya. Sebagai contoh, Anda dapat menulis kondisi kebijakan untuk menentukan bahwa semua permintaan harus dikirim menggunakan SSL. Anda juga dapat menggunakan ketentuan untuk memberikan akses ke tindakan layanan jika digunakan melalui yang spesifik Layanan AWS, seperti CloudFormation. Untuk informasi selengkapnya, lihat Elemen kebijakan JSON IAM: Kondisi dalam Panduan Pengguna IAM.
- Gunakan IAM Access Analyzer untuk memvalidasi kebijakan IAM Anda untuk memastikan izin yang aman dan fungsional – IAM Access Analyzer memvalidasi kebijakan baru dan yang sudah ada sehingga kebijakan tersebut mematuhi bahasa kebijakan IAM (JSON) dan praktik terbaik IAM. IAM Access Analyzer menyediakan lebih dari 100 pemeriksaan kebijakan dan rekomendasi yang dapat ditindaklanjuti untuk membantu Anda membuat kebijakan yang aman dan fungsional. Untuk informasi selengkapnya, lihat Validasi kebijakan dengan IAM Access Analyzer dalam Panduan Pengguna IAM.

- Memerlukan otentikasi multi-faktor (MFA) - Jika Anda memiliki skenario yang mengharuskan pengguna IAM atau pengguna root di Anda, Akun AWS aktifkan MFA untuk keamanan tambahan. Untuk meminta MFA ketika operasi API dipanggil, tambahkan kondisi MFA pada kebijakan Anda. Untuk informasi selengkapnya, lihat Amankan akses API dengan MFA dalam Panduan Pengguna IAM.

Untuk informasi selengkapnya tentang praktik terbaik dalam IAM, lihat Praktik terbaik keamanan di IAM dalam Panduan Pengguna IAM.

Menggunakan konsol Lambda

Untuk mengakses AWS Lambda konsol, Anda harus memiliki set izin minimum. Izin ini harus memungkinkan Anda untuk membuat daftar dan melihat detail tentang sumber daya Lambda di situs Anda. Akun AWS Jika Anda membuat kebijakan berbasis identitas yang lebih ketat daripada izin minimum yang diperlukan, konsol tidak akan berfungsi sebagaimana mestinya untuk entitas (pengguna atau peran) dengan kebijakan tersebut.

Anda tidak perlu mengizinkan izin konsol minimum untuk pengguna yang melakukan panggilan hanya ke AWS CLI atau AWS API. Sebagai gantinya, izinkan akses hanya ke tindakan yang sesuai dengan operasi API yang coba mereka lakukan.

Untuk contoh kebijakan yang memberikan akses minimal untuk pengembangan fungsi, lihat Memberikan pengguna akses ke Fungsi Acara. Selain API Lambda, konsol Lambda menggunakan layanan lain untuk menampilkan konfigurasi pemicu dan memungkinkan Anda menambahkan pemicu baru. Jika pengguna Anda menggunakan Lambda dengan layanan lain, mereka juga memerlukan akses ke layanan tersebut. Untuk perincian konfigurasi layanan lain dengan Lambda, lihat Memanggil Lambda dengan acara dari yang lain AWS layanan.

Perbolehkan pengguna untuk melihat izin mereka sendiri

Contoh ini menunjukkan cara membuat kebijakan yang mengizinkan pengguna IAM melihat kebijakan inline dan terkelola yang dilampirkan ke identitas pengguna mereka. Kebijakan ini mencakup izin untuk menyelesaikan tindakan ini di konsol atau menggunakan API atau secara terprogram. AWS CLI AWS

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS kebijakan terkelola untuk AWS Lambda

Kebijakan AWS terkelola adalah kebijakan mandiri yang dibuat dan dikelola oleh AWS. AWS Kebijakan terkelola dirancang untuk memberikan izin bagi banyak kasus penggunaan umum sehingga Anda dapat mulai menetapkan izin kepada pengguna, grup, dan peran.

Perlu diingat bahwa kebijakan AWS terkelola mungkin tidak memberikan izin hak istimewa paling sedikit untuk kasus penggunaan spesifik Anda karena tersedia untuk digunakan semua pelanggan. AWS Kami menyarankan Anda untuk mengurangi izin lebih lanjut dengan menentukan kebijakan yang dikelola pelanggan yang khusus untuk kasus penggunaan Anda.

Anda tidak dapat mengubah izin yang ditentukan dalam kebijakan AWS terkelola. Jika AWS memperbarui izin yang ditentukan dalam kebijakan AWS terkelola, pembaruan akan memengaruhi semua identitas utama (pengguna, grup, dan peran) yang dilampirkan kebijakan tersebut. AWS kemungkinan besar akan memperbarui kebijakan AWS terkelola saat baru Layanan AWS diluncurkan atau operasi API baru tersedia untuk layanan yang ada.

Untuk informasi selengkapnya, lihat Kebijakan terkelola AWS dalam Panduan Pengguna IAM.

Topik

- AWS kebijakan terkelola: AWS Lambda_FullAccess
- AWS kebijakan terkelola: AWS Lambda_ReadOnlyAccess
- AWS kebijakan terkelola: AWSLambdaBasicExecutionRole
- AWS kebijakan terkelola: AWSLambdaBasicDurableExecutionRolePolicy
- AWS kebijakan terkelola: AWSLambdaDynamoDBExecutionRole
- AWS kebijakan terkelola: AWSLambdaENIManagementAccess
- AWS kebijakan terkelola: AWSLambdaInvocation-DynamoDB
- AWS kebijakan terkelola: AWSLambdaKinesisExecutionRole
- AWS kebijakan terkelola: AWSLambdaMSKExecutionRole
- AWS kebijakan terkelola: AWSLambdaRole
- AWS kebijakan terkelola: AWSLambdaSQSQueueExecutionRole
- AWS kebijakan terkelola: AWSLambdaVPCAccessExecutionRole
- AWS kebijakan terkelola: AWSLambdaManagedEC2ResourceOperator
- AWS kebijakan terkelola: AWSLambdaServiceRolePolicy
- AWS kebijakan terkelola: AWSLambdaNetworkConnectorOperatorPolicy
- AWS kebijakan terkelola: AWSLambdaInvokeWebFunctionEndpointAccess
- Lambda memperbarui ke AWS kebijakan terkelola

AWS kebijakan terkelola: AWS Lambda_FullAccess

Kebijakan ini memberikan akses penuh ke tindakan Lambda. Ini juga memberikan izin ke AWS layanan lain yang digunakan untuk mengembangkan dan memelihara sumber daya Lambda.

Anda dapat melampirkan `AWSLambda_FullAccess` kebijakan ke pengguna, grup, dan peran Anda.

Detail izin

Kebijakan ini mencakup izin berikut:

- `lambda`— Memungkinkan kepala sekolah akses penuh ke Lambda.
- `cloudformation`— Memungkinkan kepala sekolah untuk menggambarkan AWS CloudFormation tumpukan dan daftar sumber daya di tumpukan tersebut.
- `cloudwatch`— Memungkinkan kepala sekolah untuk mencantumkan metrik Amazon dan mendapatkan CloudWatch data metrik.
- `ec2`— Memungkinkan kepala sekolah untuk menggambarkan kelompok keamanan, subnet, dan VPC.
- `iam`— Memungkinkan prinsipal untuk mendapatkan kebijakan, versi kebijakan, peran, kebijakan peran, kebijakan peran terlampir, dan daftar peran. Kebijakan ini juga memungkinkan kepala sekolah untuk meneruskan peran ke Lambda. `PassRole`Izin digunakan saat Anda menetapkan peran eksekusi ke fungsi. `CreateServiceLinkedRole`Izin digunakan saat membuat peran terkait layanan.
- `kms`— Memungkinkan prinsipal untuk membuat daftar alias dan menjelaskan kunci untuk enkripsi volume.
- `logs`— Memungkinkan prinsipal untuk mendeskripsikan aliran log, mendapatkan peristiwa log, memfilter peristiwa log, dan untuk memulai dan menghentikan sesi Live Tail.
- `states`— Memungkinkan kepala sekolah untuk mendeskripsikan dan membuat daftar AWS Step Functions mesin negara.
- `tag`— Memungkinkan prinsipal untuk mendapatkan sumber daya berdasarkan tag mereka.
- `xray`— Memungkinkan kepala sekolah untuk mendapatkan ringkasan AWS X-Ray jejak dan mengambil daftar jejak yang ditentukan oleh ID.

Untuk informasi selengkapnya tentang kebijakan ini, termasuk dokumen kebijakan JSON dan versi kebijakan, lihat AWS Lambda_FullAccess di Panduan Referensi Kebijakan AWS Terkelola.

AWS kebijakan terkelola: AWS Lambda_ReadOnlyAccess

Kebijakan ini memberikan akses hanya-baca ke sumber daya Lambda dan AWS layanan lain yang digunakan untuk mengembangkan dan memelihara sumber daya Lambda.

Anda dapat melampirkan `AWSLambda_ReadOnlyAccess` kebijakan ke pengguna, grup, dan peran Anda.

Detail izin

Kebijakan ini mencakup izin berikut:

- `lambda`— Memungkinkan kepala sekolah untuk mendapatkan dan membuat daftar semua sumber daya.
- `cloudformation`— Memungkinkan kepala sekolah untuk mendeskripsikan dan daftar AWS CloudFormation tumpukan dan daftar sumber daya di tumpukan tersebut.
- `cloudwatch`— Memungkinkan kepala sekolah untuk mencantumkan metrik Amazon dan mendapatkan CloudWatch data metrik.
- `ec2`— Memungkinkan kepala sekolah untuk menggambarkan kelompok keamanan, subnet, dan VPC.
- `iam`— Memungkinkan prinsipal untuk mendapatkan kebijakan, versi kebijakan, peran, kebijakan peran, kebijakan peran terlampir, dan daftar peran.
- `kms`— Memungkinkan kepala sekolah untuk daftar alias.
- `logs`— Memungkinkan prinsipal untuk mendeskripsikan aliran log, mendapatkan peristiwa log, memfilter peristiwa log, dan untuk memulai dan menghentikan sesi Live Tail.
- `states`— Memungkinkan kepala sekolah untuk mendeskripsikan dan membuat daftar AWS Step Functions mesin negara.
- `tag`— Memungkinkan prinsipal untuk mendapatkan sumber daya berdasarkan tag mereka.
- `xray`— Memungkinkan kepala sekolah untuk mendapatkan ringkasan AWS X-Ray jejak dan mengambil daftar jejak yang ditentukan oleh ID.

Untuk informasi selengkapnya tentang kebijakan ini, termasuk dokumen kebijakan JSON dan versi kebijakan, lihat AWS Lambda_ReadOnlyAccess di Panduan Referensi Kebijakan AWS Terkelola.

AWS kebijakan terkelola: AWSLambdaBasicExecutionRole

Kebijakan ini memberikan izin untuk mengunggah log ke CloudWatch Log.

Anda dapat melampirkan `AWSLambdaBasicExecutionRole` kebijakan ke pengguna, grup, dan peran Anda.

Untuk informasi selengkapnya tentang kebijakan ini, termasuk dokumen kebijakan JSON dan versi kebijakan, lihat AWSLambdaBasicExecutionRoledi Panduan Referensi Kebijakan AWS Terkelola.

AWS kebijakan terkelola: AWSLambdaBasicDurableExecutionRolePolicy

Kebijakan ini memberikan izin menulis ke CloudWatch Log dan read/write izin untuk API eksekusi tahan lama yang digunakan oleh fungsi tahan lama Lambda. Kebijakan ini memberikan izin penting yang diperlukan untuk fungsi tahan lama Lambda, yang menggunakan API eksekusi tahan lama untuk mempertahankan kemajuan dan mempertahankan status di seluruh pemanggilan fungsi.

Anda dapat melampirkan `AWSLambdaBasicDurableExecutionRolePolicy` kebijakan ke pengguna, grup, dan peran Anda.

Detail izin

Kebijakan ini mencakup izin berikut:

- `logs`— Memungkinkan prinsipal untuk membuat grup log dan aliran log, dan menulis peristiwa log ke Log. CloudWatch
- `lambda`- Memungkinkan kepala sekolah untuk memeriksa status eksekusi yang tahan lama dan mengambil status eksekusi yang tahan lama untuk fungsi Lambda yang tahan lama.

Untuk melihat detail selengkapnya tentang kebijakan, termasuk versi terbaru dari dokumen kebijakan JSON, lihat AWSLambdaBasicDurableExecutionRolePolicydi Panduan Referensi Kebijakan AWS Terkelola.

AWS kebijakan terkelola: AWSLambdaDynamoDBExecutionRole

Kebijakan ini memberikan izin untuk membaca catatan dari aliran Amazon DynamoDB dan menulis ke Log. CloudWatch

Anda dapat melampirkan `AWSLambdaDynamoDBExecutionRole` kebijakan ke pengguna, grup, dan peran Anda.

Untuk informasi selengkapnya tentang kebijakan ini, termasuk dokumen kebijakan JSON dan versi kebijakan, lihat AWSLambdaDynamoDBExecutionRoledi Panduan Referensi Kebijakan AWS Terkelola.

AWS kebijakan terkelola: AWSLambdaENIManagementAccess

Kebijakan ini memberikan izin untuk membuat, mendeskripsikan, dan menghapus antarmuka jaringan elastis yang digunakan oleh fungsi Lambda VPC-enabled .

Anda dapat melampirkan `AWSLambdaENIManagementAccess` kebijakan ke pengguna, grup, dan peran Anda.

Untuk informasi selengkapnya tentang kebijakan ini, termasuk dokumen kebijakan JSON dan versi kebijakan, lihat AWSLambdaENIManagementAccessdi Panduan Referensi Kebijakan AWS Terkelola.

AWS kebijakan terkelola: AWSLambdaInvocation-DynamoDB

Kebijakan ini memberikan akses baca ke Amazon DynamoDB Streams.

Anda dapat melampirkan `AWSLambdaInvocation-DynamoDB` kebijakan ke pengguna, grup, dan peran Anda.

Untuk informasi selengkapnya tentang kebijakan ini, termasuk dokumen kebijakan JSON dan versi kebijakan, lihat AWSLambdaInvocation-DynamoDBdi Panduan Referensi Kebijakan AWS Terkelola.

AWS kebijakan terkelola: AWSLambdaKinesisExecutionRole

Kebijakan ini memberikan izin untuk membaca peristiwa dari aliran data Amazon Kinesis dan menulis ke Log. CloudWatch

Anda dapat melampirkan `AWSLambdaKinesisExecutionRole` kebijakan ke pengguna, grup, dan peran Anda.

Untuk informasi selengkapnya tentang kebijakan ini, termasuk dokumen kebijakan JSON dan versi kebijakan, lihat AWSLambdaKinesisExecutionRoledi Panduan Referensi Kebijakan AWS Terkelola.

AWS kebijakan terkelola: AWSLambdaMSKExecutionRole

Kebijakan ini memberikan izin untuk membaca dan mengakses catatan dari klaster Amazon Managed Streaming for Apache Kafka, mengelola antarmuka jaringan elastis, dan menulis ke Log. CloudWatch

Anda dapat melampirkan `AWSLambdaMSKExecutionRole` kebijakan ke pengguna, grup, dan peran Anda.

Untuk informasi selengkapnya tentang kebijakan ini, termasuk dokumen kebijakan JSON dan versi kebijakan, lihat AWSLambdaMSKExecutionRoledi Panduan Referensi Kebijakan AWS Terkelola.

AWS kebijakan terkelola: AWSLambdaRole

Kebijakan ini memberikan izin untuk menjalankan fungsi Lambda.

Anda dapat melampirkan `AWSLambdaRole` kebijakan ke pengguna, grup, dan peran Anda.

Untuk informasi selengkapnya tentang kebijakan ini, termasuk dokumen kebijakan JSON dan versi kebijakan, lihat AWSLambdaRoledi Panduan Referensi Kebijakan AWS Terkelola.

AWS kebijakan terkelola: AWSLambdaSQSQueueExecutionRole

Kebijakan ini memberikan izin untuk membaca dan menghapus pesan dari antrian Layanan Antrian Sederhana Amazon, dan memberikan izin menulis ke Log. CloudWatch

Anda dapat melampirkan `AWSLambdaSQSQueueExecutionRole` kebijakan ke pengguna, grup, dan peran Anda.

Untuk informasi selengkapnya tentang kebijakan ini, termasuk dokumen kebijakan JSON dan versi kebijakan, lihat AWSLambdaSQSQueueExecutionRoledi Panduan Referensi Kebijakan AWS Terkelola.

AWS kebijakan terkelola: AWSLambdaVPCAccessExecutionRole

Kebijakan ini memberikan izin untuk mengelola antarmuka jaringan elastis dalam Amazon Virtual Private Cloud dan menulis ke Log. CloudWatch

Anda dapat melampirkan `AWSLambdaVPCAccessExecutionRole` kebijakan ke pengguna, grup, dan peran Anda.

Untuk informasi selengkapnya tentang kebijakan ini, termasuk dokumen kebijakan JSON dan versi kebijakan, lihat AWSLambdaVPCAccessExecutionRoledi Panduan Referensi Kebijakan AWS Terkelola.

AWS kebijakan terkelola: AWSLambdaManagedEC2ResourceOperator

Kebijakan ini memungkinkan pengelolaan instans Amazon Elastic Compute Cloud otomatis dan CloudWatch pencatatan Amazon untuk penyedia kapasitas Lambda. Ini memberikan izin kepada Lambda untuk melakukan operasi siklus hidup instance dan menulis log atas nama Anda.

Anda dapat melampirkan `AWSLambdaManagedEC2ResourceOperator` kebijakan ke pengguna, grup, dan peran Anda.

Detail izin

Kebijakan ini mencakup izin berikut:

- `ec2:RunInstances`— Memungkinkan Lambda meluncurkan instans Amazon EC2 baru dengan syarat ec2: ManagedResourceOperator sama dengan scaler.lambda.amazonaws.com dan membatasi penggunaan AMI hanya untuk gambar. Amazon-owned
- `ec2:DescribeInstances`dan `ec2:DescribeInstanceStatus` — Memungkinkan Lambda untuk memantau status instance dan mengambil informasi instance.
- `ec2:CreateTags`— Memungkinkan Lambda menandai sumber daya Amazon EC2 untuk tujuan manajemen dan identifikasi.
- `ec2:DescribeAvailabilityZones`— Memungkinkan Lambda untuk melihat zona yang tersedia misalnya keputusan penempatan.
- `ec2:DescribeCapacityReservations`— Memungkinkan Lambda untuk memeriksa reservasi kapasitas untuk penempatan instans yang optimal.
- `ec2:DescribeInstanceTypes`dan `ec2:DescribeInstanceTypeOfferings` — Memungkinkan Lambda untuk meninjau jenis instans yang tersedia dan penawarannya.
- `ec2:DescribeSubnets`— Memungkinkan Lambda untuk memeriksa konfigurasi subnet untuk perencanaan jaringan.
- `ec2:DescribeVpcEncryptionControls`— Memungkinkan Lambda untuk memeriksa pengaturan kontrol enkripsi VPC untuk konfigurasi misalnya.
- `ec2:DescribeSecurityGroups`— Memungkinkan Lambda untuk mengambil informasi grup keamanan untuk konfigurasi antarmuka jaringan.
- `ec2:CreateNetworkInterface`— Memungkinkan Lambda untuk membuat antarmuka jaringan dan mengelola subnet dan asosiasi grup keamanan.
- `ec2:AttachNetworkInterface`— Memungkinkan Lambda melampirkan antarmuka jaringan ke instans Amazon EC2 dengan kondisi yang sama dengan scaler.lambda.amazonaws.com. `ec2:ManagedResourceOperator`
- `logs:CreateLogGroup`— Memungkinkan Lambda membuat grup CloudWatch log Log.
- `logs:CreateLogStream`dan `logs:PutLogEvents` — Memungkinkan Lambda membuat aliran log dan menulis peristiwa log ke Log. CloudWatch

Untuk informasi selengkapnya tentang kebijakan ini, termasuk dokumen kebijakan JSON dan versi kebijakan, lihat AWSLambdaManagedEC2ResourceOperatordi Panduan Referensi Kebijakan AWS Terkelola.

AWS kebijakan terkelola: AWSLambdaServiceRolePolicy

Kebijakan ini dilampirkan ke peran terkait layanan yang diberi nama AWSServiceRoleForLambda untuk mengizinkan Lambda 1) menghentikan instans yang dikelola sebagai bagian dari penyedia kapasitas Lambda dan 2) menghapus Antarmuka Jaringan Elastis (ENI) yang dikelola sebagai bagian dari konektor jaringan Lambda.

Detail izin

Kebijakan ini mencakup izin berikut:

- `ec2:TerminateInstances`— Memungkinkan Lambda untuk menghentikan instans EC2 dengan kondisi bahwa ec2: sama dengan scaler.lambda.amazonaws.com. ManagedResourceOperator
- `ec2:DescribeInstanceStatus`dan `ec2:DescribeInstances` — Memungkinkan Lambda untuk menggambarkan instans EC2.
- `ec2:DescribeNetworkInterfaces`,`ec2:DescribeSubnets`, dan `ec2:DescribeSecurityGroups` — Memungkinkan Lambda untuk menggambarkan antarmuka jaringan, subnet, dan kelompok keamanan.
- `ec2:DeleteNetworkInterface`— Memungkinkan Lambda untuk menghapus ENI dengan syarat ec2: sama dengan network-connectors.lambda.amazonaws.com. ManagedResourceOperator

Untuk informasi selengkapnya tentang kebijakan ini, lihat Menggunakan peran terkait layanan untuk Lambda.

AWS kebijakan terkelola: AWSLambdaNetworkConnectorOperatorPolicy

Kebijakan ini memberikan izin untuk membuat dan mengelola resource elastic network interface (ENI) yang dikelola oleh Lambda Network Connector.

Anda dapat melampirkan `AWSLambdaNetworkConnectorOperatorPolicy` kebijakan ke pengguna, grup, dan peran Anda.

Detail izin

Kebijakan ini mencakup izin berikut:

- `ec2:CreateNetworkInterface`— Memberikan izin untuk membuat antarmuka jaringan di subnet apa pun dengan grup keamanan terkait. Untuk sumber daya antarmuka

jaringan, hanya tag dengan kunci `aws:lambda:networkConnectorName` dan `aws:lambda:networkConnectorId` diizinkan.

- `ec2:CreateTags`— Memberikan izin untuk menandai antarmuka jaringan pada waktu pembuatan. Izin ini hanya berlaku ketika tindakan buat `CreateNetworkInterface` dan operator sumber daya terkelola`network-connectors.lambda.amazonaws.com`.

Untuk informasi selengkapnya tentang kebijakan ini, termasuk dokumen kebijakan JSON dan versi kebijakan, lihat AWSLambdaNetworkConnectorOperatorPolicydi Panduan Referensi Kebijakan AWS Terkelola.

AWS kebijakan terkelola: AWSLambdaInvokeWebFunctionEndpointAccess

Kebijakan ini memberikan izin untuk memanggil titik akhir fungsi web Lambda.

Anda dapat melampirkan `AWSLambdaInvokeWebFunctionEndpointAccess` kebijakan ke pengguna, grup, dan peran Anda.

Untuk informasi selengkapnya tentang kebijakan ini, termasuk dokumen kebijakan JSON dan versi kebijakan, lihat AWSLambdaInvokeWebFunctionEndpointAccessdi Panduan Referensi Kebijakan AWS Terkelola.

Lambda memperbarui ke AWS kebijakan terkelola

Ubah	Deskripsi	Date
AWSLambdaInvokeWebFunctionEndpointAccess – Kebijakan baru	Lambda menambahkan kebijakan terkelola baru untuk memberikan izin untuk memanggil titik akhir fungsi web Lambda.	Oktober 1, 2026
AWSLambdaNetworkConnectorOperatorPolicy – Kebijakan baru	Lambda menambahkan kebijakan terkelola baru untuk memberikan izin untuk membuat dan mengelola sumber daya ENI yang dikelola oleh Konektor Jaringan Lambda.	Juni 19, 2026

Ubah	Deskripsi	Date
AWSLambdaManagedEC2ResourceOperator— Ubah	Lambda memperbarui `AWSLambdaManagedEC2ResourceOperator` kebijakan untuk mengizinkan `logs:CreateLogGroup` , `logs:CreateLogStream` , dan `logs:PutLogEvents` tindakan. Izin ini memungkinkan Lambda untuk menulis log ke Log CloudWatch .	Juni 10, 2026
AWSLambdaManagedEC2ResourceOperator— Ubah	Lambda memperbarui `AWSLambdaManagedEC2ResourceOperator` kebijakan untuk mengizinkan tindakan. `ec2:DescribeVpcEncryptionControls` Izin ini memungkinkan Lambda untuk memeriksa pengaturan kontrol enkripsi VPC untuk konfigurasi misalnya.	April 23, 2026
AWSLambdaManagedEC2ResourceOperator – Kebijakan baru	Lambda menambahkan kebijakan terkelola baru untuk mengaktifkan manajemen instans Amazon EC2 otomatis untuk penyedia kapasitas Lambda, yang memungkinkan layanan scaler untuk melakukan operasi siklus hidup instans.	November 30, 2025

Ubah	Deskripsi	Date
AWSLambdaServiceRolePolicy – Kebijakan baru	Lambda menambahkan kebijakan terkelola baru untuk peran terkait layanan guna memungkinkan Lambda menghentikan instans yang dikelola sebagai bagian dari penyedia kapasitas Lambda.	November 30, 2025
AWS Lambda_FullAccess - Ubah	Lambda memperbarui `AWSLambda_FullAccess` kebijakan untuk mengizinkan `kms:DescribeKey` dan `iam:CreateServiceLinkedRole` tindakan.	November 30, 2025
AWSLambdaBasicDurableExecutionRolePolicy— Kebijakan terkelola baru	Lambda merilis kebijakan terkelola baru `AWSLambdaBasicDurableExecutionRolePolicy` yang menyediakan izin menulis ke CloudWatch Log dan read/write izin untuk API eksekusi tahan lama yang digunakan oleh fungsi tahan lama Lambda.	Desember 1, 2025
AWS Lambda_ReadOnlyAccess dan AWS Lambda_FullAccess — Ubah	Lambda memperbarui `AWSLambda_FullAccess` kebijakan `AWSLambda_ReadOnlyAccess` dan untuk mengizinkan `logs:StartLiveTail` dan `logs:StopLiveTail` tindakan.	Maret 17, 2025

Ubah	Deskripsi	Date
AWSLambdaVPCAccess ExecutionRole— Ubah	Lambda memperbarui `AWSLambdaVPCAccess ExecutionRole` kebijakan untuk mengizinkan tindakan. `ec2:DescribeSubnets`	Januari 5, 2024
AWS Lambda_ReadOnlyAcc ess - Ubah	Lambda memperbarui `AWSLambda_ReadOnly Access` kebijakan untuk mengizinkan prinsipal membuat daftar tumpukan. CloudFormation	Juli 27, 2023
AWS Lambda mulai melacak perubahan	AWS Lambda mulai melacak perubahan untuk kebijakan yang AWS dikelola.	Juli 27, 2023

Pemecahan masalah AWS Lambda identitas dan akses

Gunakan informasi berikut untuk membantu Anda mendiagnosis dan memperbaiki masalah umum yang mungkin Anda temukan saat bekerja dengan Lambda dan IAM.

Topik

- Saya tidak diotorisasi untuk melakukan tindakan di Lambda
- Saya tidak berwenang untuk melakukan iam: PassRole
- Saya ingin mengizinkan orang di luar saya Akun AWS untuk mengakses sumber daya Lambda saya

Saya tidak diotorisasi untuk melakukan tindakan di Lambda

Jika Anda menerima pesan kesalahan bahwa Anda tidak memiliki otorisasi untuk melakukan tindakan, kebijakan Anda harus diperbarui agar Anda dapat melakukan tindakan tersebut.

Contoh kesalahan berikut terjadi ketika pengguna IAM `mateojackson` mencoba menggunakan konsol untuk melihat detail tentang suatu sumber daya *`my-example-widget`* rekaan, tetapi tidak memiliki izin `lambda:`*`GetWidget`* rekaan.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 lambda:GetWidget on resource: my-example-widget
```

Dalam hal ini, kebijakan untuk pengguna `mateojackson` harus diperbarui untuk mengizinkan akses ke sumber daya *`my-example-widget`* dengan menggunakan tindakan `lambda:`*`GetWidget`*.

Jika Anda memerlukan bantuan, hubungi AWS administrator Anda. Administrator Anda adalah orang yang memberi Anda kredensial masuk.

Saya tidak berwenang untuk melakukan iam: PassRole

Jika Anda menerima kesalahan bahwa Anda tidak diizinkan untuk melakukan `iam:PassRole` tindakan, kebijakan Anda harus diperbarui agar Anda dapat meneruskan peran ke Lambda.

Beberapa Layanan AWS memungkinkan Anda untuk meneruskan peran yang ada ke layanan tersebut alih-alih membuat peran layanan baru atau peran terkait layanan. Untuk melakukannya, Anda harus memiliki izin untuk meneruskan peran ke layanan.

Contoh kesalahan berikut terjadi saat pengguna IAM bernama `marymajor` mencoba menggunakan konsol untuk melakukan tindakan di Lambda. Namun, tindakan tersebut memerlukan layanan untuk mendapatkan izin yang diberikan oleh peran layanan. Mary tidak memiliki izin untuk meneruskan peran tersebut pada layanan.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

Dalam kasus ini, kebijakan Mary harus diperbarui agar dia mendapatkan izin untuk melakukan tindakan `iam:PassRole` tersebut.

Jika Anda memerlukan bantuan, hubungi AWS administrator Anda. Administrator Anda adalah orang yang memberi Anda kredensial masuk.

Saya ingin mengizinkan orang di luar saya Akun AWS untuk mengakses sumber daya Lambda saya

Anda dapat membuat peran yang dapat digunakan pengguna di akun lain atau orang-orang di luar organisasi Anda untuk mengakses sumber daya Anda. Anda dapat menentukan siapa saja yang

dipercaya untuk mengambil peran tersebut. Untuk layanan yang mendukung kebijakan berbasis sumber daya atau daftar kontrol akses (ACL), Anda dapat menggunakan kebijakan tersebut untuk memberi orang akses ke sumber daya Anda.

Untuk mempelajari lebih lanjut, lihat hal berikut:

- Untuk mempelajari apakah Lambda mendukung fitur-fitur ini, lihat Bagaimana AWS Lambda bekerja dengan IAM.
- Untuk mempelajari cara menyediakan akses ke sumber daya Anda di seluruh sumber daya Akun AWS yang Anda miliki, lihat Menyediakan akses ke pengguna IAM di pengguna lain Akun AWS yang Anda miliki di Panduan Pengguna IAM.
- Untuk mempelajari cara menyediakan akses ke sumber daya Anda kepada pihak ketiga Akun AWS, lihat Menyediakan akses yang Akun AWS dimiliki oleh pihak ketiga dalam Panduan Pengguna IAM.
- Untuk mempelajari cara memberikan akses melalui federasi identitas, lihat Menyediakan akses ke pengguna terautentikasi eksternal (federasi identitas) dalam Panduan Pengguna IAM.
- Untuk mempelajari perbedaan antara menggunakan peran dan kebijakan berbasis sumber daya untuk akses lintas akun, lihat Akses sumber daya lintas akun di IAM di Panduan Pengguna IAM.

Buat strategi tata kelola untuk fungsi dan lapisan Lambda

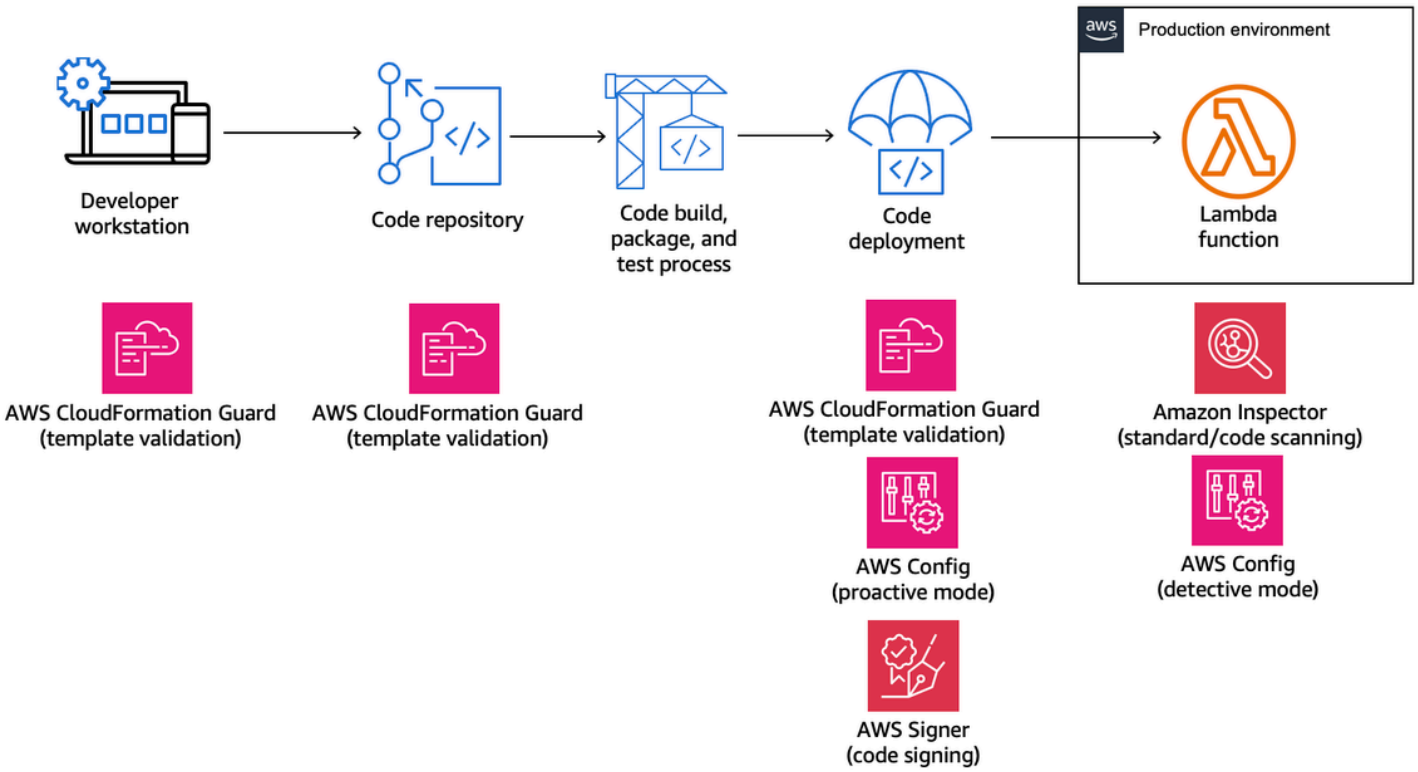
Untuk membangun dan menerapkan aplikasi cloud-native tanpa server, Anda harus memungkinkan kelincahan dan kecepatan untuk dipasarkan dengan tata kelola dan pagar pembatas yang sesuai. Anda menetapkan prioritas tingkat bisnis, mungkin menekankan kelincahan sebagai prioritas utama, atau sebagai alternatif menekankan penghindaran risiko melalui tata kelola, pagar pembatas, dan kontrol. Secara realistis, Anda tidak akan memiliki strategi "either/or" tetapi strategi “dan” yang menyeimbangkan kelincahan dan pagar pembatas dalam siklus pengembangan perangkat lunak Anda. Di mana pun persyaratan ini berada dalam siklus hidup perusahaan Anda, kemampuan tata kelola cenderung menjadi persyaratan implementasi dalam proses dan rantai alat Anda.

Berikut adalah beberapa contoh kontrol tata kelola yang mungkin diterapkan organisasi untuk Lambda:

- Fungsi lambda tidak boleh diakses publik.
- Fungsi lambda harus dilampirkan ke VPC.
- Fungsi lambda tidak boleh menggunakan runtime yang tidak digunakan lagi.

- Fungsi lambda harus ditandai dengan satu set tag yang diperlukan.
- Lapisan Lambda tidak boleh diakses di luar organisasi.
- Fungsi lambda dengan grup keamanan terlampir harus memiliki tag yang cocok antara fungsi dan grup keamanan.
- Fungsi lambda dengan lapisan terlampir harus menggunakan versi yang disetujui
- Variabel lingkungan lambda harus dienkripsi saat diam dengan kunci yang dikelola pelanggan.

Diagram berikut adalah contoh strategi tata kelola mendalam yang menerapkan kontrol dan kebijakan di seluruh proses pengembangan dan penyebaran perangkat lunak:



Topik berikut menjelaskan cara menerapkan kontrol untuk mengembangkan dan menerapkan fungsi Lambda di organisasi Anda, baik untuk startup maupun perusahaan. Organisasi Anda mungkin sudah memiliki alat. Topik berikut mengambil pendekatan modular untuk kontrol ini, sehingga Anda dapat memilih dan memilih komponen yang benar-benar Anda butuhkan. Topik meliputithe section called “Kontrol proaktif dengan Guard”,the section called “Pemindaian kode”,the section called “Penandatanganan kode”, danthe section called “Observabilitas”.

Topik

- Kontrol proaktif untuk Lambda dengan AWS CloudFormation Guard

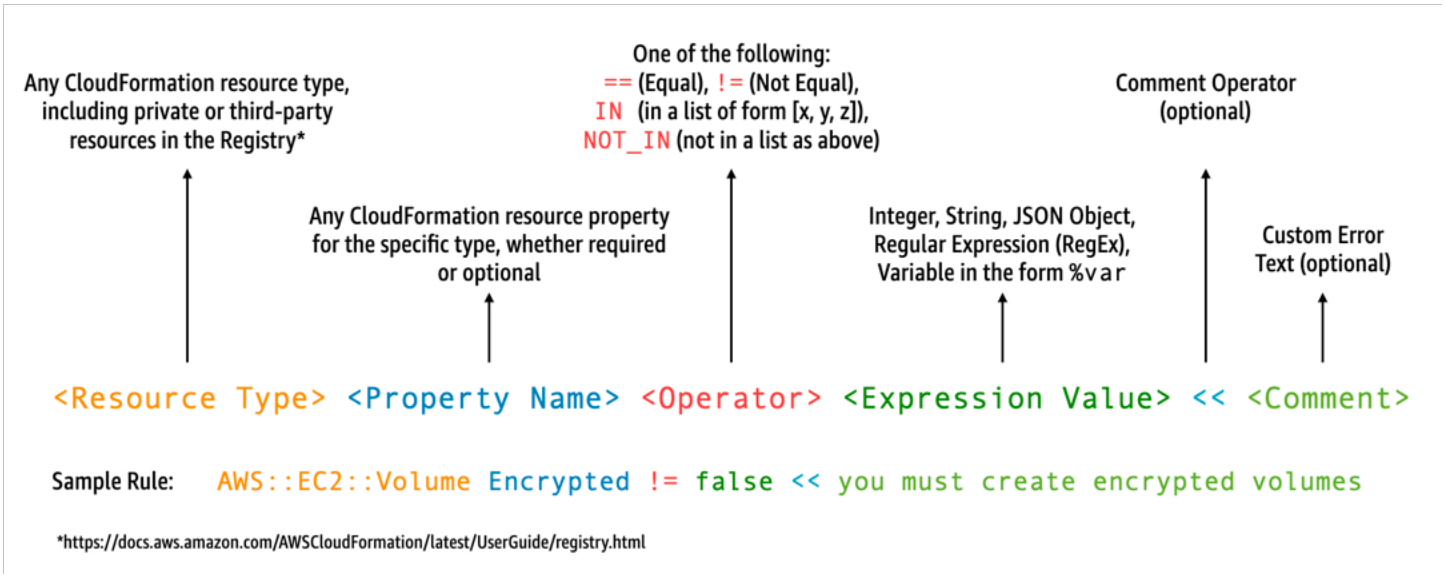
- Menerapkan kontrol pencegahan untuk Lambda dengan AWS Config
- Mendeteksi penerapan dan konfigurasi Lambda yang tidak sesuai dengan AWS Config
- Penandatanganan kode lambda dengan AWS Signer
- Otomatiskan penilaian keamanan untuk Lambda dengan Amazon Inspector
- Menerapkan observabilitas untuk keamanan dan kepatuhan Lambda

Kontrol proaktif untuk Lambda dengan AWS CloudFormation Guard

AWS CloudFormation Guardadalah alat evaluasi kebijakan sebagai kode sumber terbuka, tujuan umum. Ini dapat digunakan untuk tata kelola dan kepatuhan preventif dengan memvalidasi template Infrastructure as Code (IaC) dan komposisi layanan terhadap aturan kebijakan. Aturan-aturan ini dapat disesuaikan berdasarkan persyaratan tim atau organisasi Anda. Untuk fungsi Lambda, aturan Guard dapat digunakan untuk mengontrol pembuatan sumber daya dan pembaruan konfigurasi dengan menentukan pengaturan properti yang diperlukan yang diperlukan saat membuat atau memperbarui fungsi Lambda.

Administrator kepatuhan menentukan daftar kontrol dan kebijakan tata kelola yang diperlukan untuk menerapkan dan memperbarui fungsi Lambda. Administrator platform menerapkan kontrol dalam CI/CD pipeline, sebagai webhook validasi pra-komit dengan repositori kode, dan menyediakan pengembang dengan alat baris perintah untuk memvalidasi template dan kode di workstation lokal. Pengembang membuat kode, memvalidasi template dengan alat baris perintah, dan kemudian mengkomit kode ke repositori, yang kemudian secara otomatis divalidasi melalui CI/CD pipeline sebelum penerapan ke AWS lingkungan.

Dengan Guard, Anda dapat menulis aturan dan menerapkan kontrol Anda dengan bahasa khusus domain sebagai berikut.



Misalnya, Anda ingin memastikan bahwa pengembang hanya memilih runtime terbaru. Anda dapat menentukan dua kebijakan yang berbeda, satu untuk mengidentifikasi runtime yang sudah usang dan satu lagi untuk mengidentifikasi runtime yang akan segera dihentikan. Untuk melakukan ini, Anda dapat menulis `etc/rules.guard` file berikut:

```
let lambda_functions = Resources.*[
    Type == "AWS::Lambda::Function"
]

rule lambda_already_deprecated_runtime when %lambda_functions !empty {
    %lambda_functions {
        Properties {
            when Runtime exists {
                Runtime !in ["dotnetcore3.1", "nodejs12.x", "python3.6", "python2.7",
 "dotnet5.0", "dotnetcore2.1", "ruby2.5", "nodejs10.x", "nodejs8.10", "nodejs4.3",
 "nodejs6.10", "dotnetcore1.0", "dotnetcore2.0", "nodejs4.3-edge", "nodejs"] <<Lambda
 function is using a deprecated runtime.>>
            }
        }
    }
}

rule lambda_soon_to_be_deprecated_runtime when %lambda_functions !empty {
    %lambda_functions {
        Properties {
            when Runtime exists {
                Runtime !in ["nodejs16.x", "nodejs14.x", "python3.7", "java8",
 "dotnet7", "go1.x", "ruby2.7", "provided"] <<Lambda function is using a runtime that
 is targeted for deprecation.>>
            }
        }
    }
}
```

Sekarang misalkan Anda menulis `iac/lambda.yaml` CloudFormation template berikut yang mendefinisikan fungsi Lambda:

```
  Fn:
    Type: AWS::Lambda::Function
    Properties:
      Runtime: python3.7
      CodeUri: src
      Handler: fn.handler
      Role: !GetAtt FnRole.Arn
      Layers:
        - arn:aws:lambda:us-east-1:111122223333:layer:LambdaInsightsExtension:35
```

Setelah menginstal util itas Guard, validasi template Anda:

```
cfn-guard validate --rules etc/rules.guard --data iac/lambda.yaml
```

Keluaran terlihat seperti ini:

```
lambda.yaml Status = FAIL
FAILED rules
rules.guard/lambda_soon_to_be_deprecated_runtime
---
Evaluating data lambda.yaml against rules rules.guard
Number of non-compliant resources 1
Resource = Fn {
  Type      = AWS::Lambda::Function
  Rule = lambda_soon_to_be_deprecated_runtime {
    ALL {
      Check =  Runtime not IN
 ["nodejs16.x","nodejs14.x","python3.7","java8","dotnet7","go1.x","ruby2.7","provided"]
 {
        ComparisonError {
          Message          = Lambda function is using a runtime that is targeted for
 deprecation.
          Error            = Check was not compliant as property [/Resources/
Fn/Properties/Runtime[L:88,C:15]] was not present in [(resolved, Path=[L:0,C:0]
 Value=["nodejs16.x","nodejs14.x","python3.7","java8","dotnet7","go1.x","ruby2.7","provided"])]
        }
          PropertyPath    = /Resources/Fn/Properties/Runtime[L:88,C:15]
          Operator        = NOT IN
          Value           = "python3.7"
          ComparedWith    =
 [["nodejs16.x","nodejs14.x","python3.7","java8","dotnet7","go1.x","ruby2.7","provided"]]
          Code:
                86.  Fn:
                87.    Type: AWS::Lambda::Function
                88.    Properties:
                89.      Runtime: python3.7
                90.      CodeUri: src
                91.      Handler: fn.handler


      }
    }
  }
}
```

Dengan Guard, pengembang Anda dapat melihat dari workstation pengembang lokal mereka bahwa mereka perlu memperbarui template untuk menggunakan runtime yang diizinkan oleh organisasi. Ini terjadi sebelum berkomitmen ke repositori kode dan kemudian gagal memeriksa dalam CI/CD pipeline. Akibatnya, pengembang Anda mendapatkan umpan balik ini tentang cara mengembangkan template yang sesuai dan mengalihkan waktu mereka untuk menulis kode yang memberikan nilai bisnis. Kontrol ini dapat diterapkan pada workstation pengembang lokal, dalam webhook validasi pra-komit, and/or dalam CI/CD pipeline sebelum penerapan.

Peringatan

Jika Anda menggunakan template AWS Serverless Application Model (AWS SAM) untuk menentukan fungsi Lambda, ketahuilah bahwa Anda perlu memperbarui aturan Guard untuk mencari jenis `AWS::Serverless::Function` sumber daya sebagai berikut.

```
let lambda_functions = Resources.*[
    Type == "AWS::Serverless::Function"
]
```

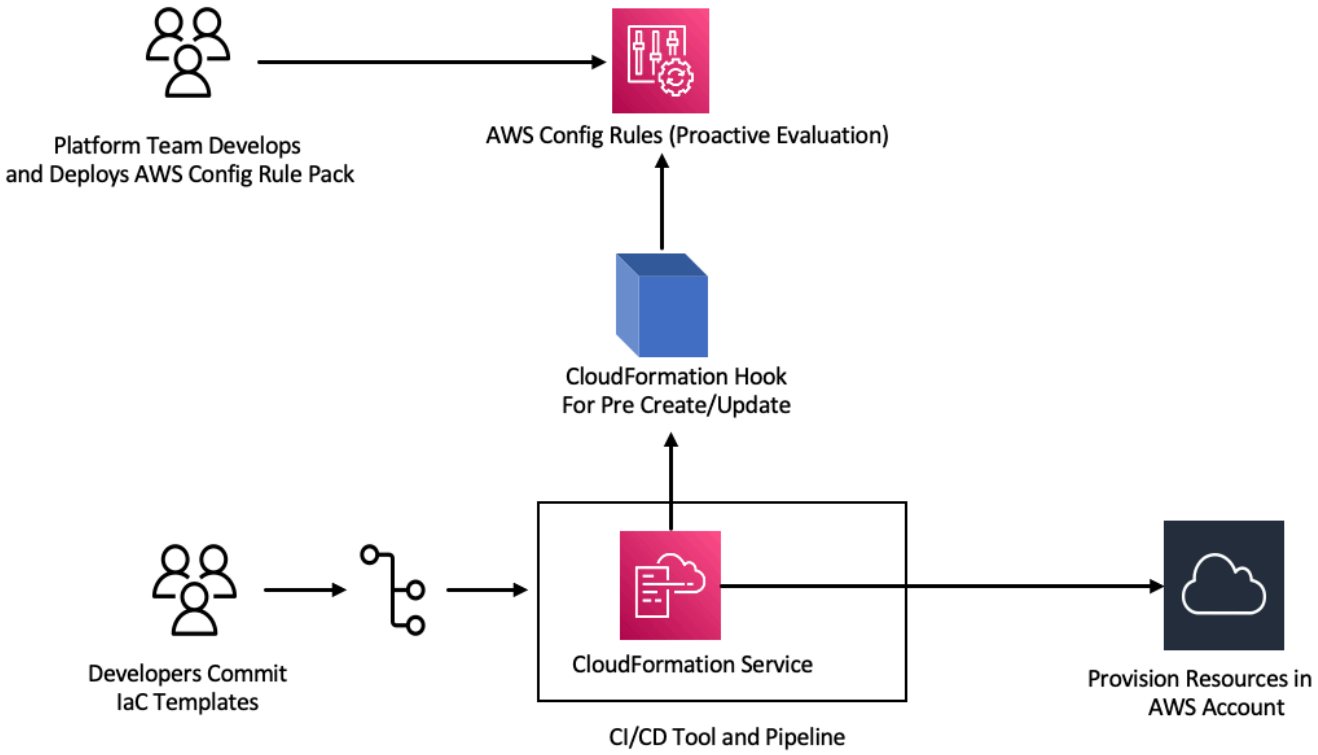
Guard juga mengharapkan properti untuk dimasukkan dalam definisi sumber daya. Sementara itu, AWS SAM template memungkinkan properti untuk ditentukan di bagian Globals yang terpisah. Properti yang didefinisikan di bagian Globals tidak divalidasi dengan aturan Guard Anda.

Sebagaimana diuraikan dalam dokumentasi pemec ahan masalah Guard, ketahuilah bahwa Guard tidak mendukung intrinsik bentuk pendek seperti `!GetAtt` atau `!Sub` dan sebaliknya memerlukan penggunaan formulir yang diperluas: dan. `Fn::GetAtt` `Fn::Sub` (Contoh sebelumnya tidak mengevaluasi properti Role, jadi intrinsik bentuk pendek digunakan untuk kesederhanaan.)

Menerapkan kontrol pencegahan untuk Lambda dengan AWS Config

Sangat penting untuk memastikan kepatuhan dalam aplikasi tanpa server Anda sedini mungkin dalam proses pengembangan. Dalam topik ini, kami membahas cara menerapkan kontrol pencegahan menggunakan AWS Config. Dengan AWS Config, Anda dapat menerapkan pemeriksaan kepatuhan lebih awal dalam proses pengembangan dan menerapkan kontrol yang sama di CI/CD pipeline Anda. Ini juga menstandarisasi kontrol Anda dalam repositori aturan yang dikelola secara terpusat sehingga Anda dapat menerapkan kontrol secara konsisten di seluruh AWS akun Anda.

Misalnya, administrator kepatuhan Anda menetapkan persyaratan untuk memastikan bahwa semua fungsi Lambda menyertakan pel AWS X-Ray acakan. Dengan AWS Config mode proaktif, Anda dapat menjalankan pemeriksaan kepatuhan pada sumber daya fungsi Lambda sebelum penerapan, mengurangi risiko penerapan fungsi Lambda yang tidak dikonfigurasi dengan benar dan menghemat waktu pengembang dengan memberi mereka umpan balik yang lebih cepat tentang infrastruktur sebagai templat kode. Berikut ini adalah visualisasi aliran untuk kontrol pencegahan dengan: AWS Config



Pertimbangkan persyaratan bahwa semua fungsi Lambda harus mengaktifkan pelacakan. Sebagai tanggapan, tim platform mengidentifikasi perlunya AWS Config aturan khusus untuk berjalan secara proaktif di semua akun. Aturan ini menandai fungsi Lambda apa pun yang tidak memiliki konfigurasi pelacakan yang X-Ray dikonfigurasi sebagai sumber daya yang tidak sesuai. Tim mengembangkan aturan, mengemasnya dalam paket kesesuaian, dan menerapkan paket kesesuaian di semua AWS akun untuk memastikan bahwa semua akun di organisasi menerapkan kontrol ini secara seragam. Anda dapat menulis aturan dalam sintaks AWS CloudFormation Guard 2.x.x, yang mengambil bentuk berikut:

```
rule name when condition { assertion }
```

Berikut ini adalah contoh aturan Guard yang memeriksa untuk memastikan fungsi Lambda telah mengaktifkan pelacakan:

```
rule lambda_tracing_check {
  when configuration.tracingConfig exists {
      configuration.tracingConfig.mode == "Active"
  }
}
```

Tim platform mengambil tindakan lebih lanjut dengan mengamanatkan bahwa setiap AWS CloudFormation penerapan memanggil pre-hook. create/update https://docs.aws.amazon.com/cloudformation-cli/latest/userguide/hooks-structure.html Mereka bertanggung jawab penuh untuk mengembangkan hook ini dan mengonfigurasi pipeline, memperkuat kontrol terpusat terhadap aturan kepatuhan dan mempertahankan aplikasi mereka yang konsisten di semua penerapan. Untuk mengembangkan, mengemas, dan mendaftarkan hook, lihat Mengem bangkan AWS CloudFormation Hooks di dokumentasi CloudFormation Command Line Interface (CFN-CLI). Anda dapat menggunakan CloudFormation CLI untuk membuat proyek hook:

```
cfn init
```

Perintah ini meminta Anda untuk beberapa informasi dasar tentang proyek hook Anda dan membuat proyek dengan file berikut di dalamnya:

```
README.md
<hook-name>.json
rpdk.log
src/handler.py
```

```
template.yml
hook-role.yaml
```

Sebagai pengembang hook, Anda perlu menambahkan jenis sumber daya target yang diinginkan dalam file `<hook-name>.json` konfigurasi. Dalam konfigurasi di bawah ini, hook dikonfigurasi untuk dieksekusi sebelum fungsi Lambda dibuat menggunakan CloudFormation. Anda dapat menambahkan penangan serupa untuk `preUpdate` dan `preDelete` tindakan juga.

```
    "handlers": {
        "preCreate": {
            "targetNames": [
                "AWS::Lambda::Function"
            ],
            "permissions": []
        }
    }
```

Anda juga perlu memastikan bahwa CloudFormation hook memiliki izin yang sesuai untuk memanggil AWS Config API. Anda dapat melakukannya dengan memperbarui file definisi peran bernama`hook-role.yaml`. File definisi peran memiliki kebijakan kepercayaan berikut secara default, yang memungkinkan CloudFormation untuk mengambil peran.

```
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Principal:
              Service:
                - hooks.cloudformation.amazonaws.com
                - resources.cloudformation.amazonaws.com
```

Untuk mengizinkan hook ini memanggil API konfigurasi, Anda harus menambahkan izin berikut ke pernyataan Kebijakan. Kemudian Anda mengirimkan proyek hook menggunakan `cfn submit` perintah, di mana CloudFormation membuat peran untuk Anda dengan izin yang diperlukan.

```
      Policies:
        - PolicyName: HookTypePolicy
          PolicyDocument:
            Version: '2012-10-17'
            Statement:
```

```
              - Effect: Allow
                Action:
                  - "config:Describe*"
                  - "config:Get*"
                  - "config:List*"
                  - "config:SelectResourceConfig"
                Resource: "*
```

Selanjutnya, Anda perlu menulis fungsi Lambda dalam `src/handler.py` file. Di dalam file ini, Anda menemukan metode bernama `preCreatepreUpdate`,, dan `preDelete` sudah dibuat ketika Anda memulai proyek. Anda bertujuan untuk menulis fungsi umum yang dapat digunakan kembali yang memanggil AWS Config `StartResourceEvaluation` API dalam mode proaktif menggunakan AWS SDK untuk Python (Boto3). Panggilan API ini mengambil properti sumber daya sebagai input dan mengevaluasi sumber daya terhadap definisi aturan.

```
def validate_lambda_tracing_config(resource_type, function_properties:
 MutableMapping[str, Any]) -> ProgressEvent:
  LOG.info("Fetching proactive data")
  config_client = boto3.client('config')
  resource_specs = {
      'ResourceId': 'MyFunction',
      'ResourceType': resource_type,
      'ResourceConfiguration': json.dumps(function_properties),
      'ResourceConfigurationSchemaType': 'CFN_RESOURCE_SCHEMA'
  }
  LOG.info("Resource Specifications:", resource_specs)
  eval_response = config_client.start_resource_evaluation(EvaluationMode='PROACTIVE',
 ResourceDetails=resource_specs, EvaluationTimeout=60)
  ResourceEvaluationId = eval_response.ResourceEvaluationId
  compliance_response =
 config_client.get_compliance_details_by_resource(ResourceEvaluationId=ResourceEvaluationId)
  LOG.info("Compliance Verification:",
 compliance_response.EvaluationResults[0].ComplianceType)
  if "NON_COMPLIANT" == compliance_response.EvaluationResults[0].ComplianceType:
      return ProgressEvent(status=OperationStatus.FAILED, message="Lambda function
 found with no tracing enabled : FAILED", errorCode=HandlerErrorCode.NonCompliant)
  else:
      return ProgressEvent(status=OperationStatus.SUCCESS, message="Lambda function
 found with tracing enabled : PASS.")
```

Sekarang Anda dapat memanggil fungsi umum dari handler untuk hook pra-buat. Berikut contoh handler:

```
@hook.handler(HookInvocationPoint.CREATE_PRE_PROVISION)
def pre_create_handler(
        session: Optional[SessionProxy],
        request: HookHandlerRequest,
        callback_context: MutableMapping[str, Any],
        type_configuration: TypeConfigurationModel
) -> ProgressEvent:
    LOG.info("Starting execution of the hook")
    target_name = request.hookContext.targetName
    LOG.info("Target Name:", target_name)
    if "AWS::Lambda::Function" == target_name:
        return validate_lambda_tracing_config(target_name,
            request.hookContext.targetModel.get("resourceProperties")
        )
    else:
        raise exceptions.InvalidRequest(f"Unknown target type: {target_name}")
```

Setelah langkah ini Anda dapat mendaftarkan hook dan mengkonfigurasinya untuk mendengarkan semua acara pembuatan AWS Lambda fungsi.

Pengembang menyiapkan infrastruktur sebagai template kode (IAC) untuk layanan mikro tanpa server menggunakan Lambda. Persiapan ini mencakup kepatuhan terhadap standar internal, diikuti dengan pengujian lokal dan memasukkan template ke repositori. Berikut contoh template IaC:

```
  MyLambdaFunction:
  Type: 'AWS::Lambda::Function'
  Properties:
    Handler: index.handler
    Role: !GetAtt LambdaExecutionRole.Arn
    FunctionName: MyLambdaFunction
    Code:
      ZipFile: |
        import json

        def handler(event, context):
            return {
                'statusCode': 200,
                'body': json.dumps('Hello World!')
            }
    Runtime: python3.14
    TracingConfig:
        Mode: PassThrough
```

```
    MemorySize: 256
    Timeout: 10
```

Sebagai bagian dari CI/CD proses, saat CloudFormation template digunakan, CloudFormation layanan akan memanggil pre- create/update hook tepat sebelum menyediakan jenis sumber daya. `AWS::Lambda::Function` Pengait menggunakan AWS Config aturan yang berjalan dalam mode proaktif untuk memverifikasi bahwa konfigurasi fungsi Lambda menyertakan konfigurasi pelacakan yang diamanatkan. Respons dari hook menentukan langkah selanjutnya.

Jika sesuai, hook menandakan keberhasilan, dan CloudFormation melanjutkan untuk menyediakan sumber daya. Jika tidak, pener CloudFormation apan tumpukan gagal, pipeline segera terhenti, dan sistem mencatat detail untuk peninjauan selanjutnya. Pemberitahuan kepatuhan dikirim ke pemangku kepentingan terkait.

Anda dapat menemukan success/fail informasi kait di CloudFormation konsol:

Stack info | Events | Resources | Outputs | Parameters | Template | Change sets

Events (19)

Search events

Timestamp	Logical ID	Status	Status reason	Hook invocations
2023-08-29 23:50:23 UTC-0500	HookTestStack	ROLLBACK_COMPLETE	-	-
2023-08-29 23:50:22 UTC-0500	LambdaExecutionRole	DELETE_COMPLETE	-	-
2023-08-29 23:50:21 UTC-0500	MyApi	DELETE_COMPLETE	-	-
2023-08-29 23:50:20 UTC-0500	LambdaExecutionRole	DELETE_IN_PROGRESS	-	-
2023-08-29 23:50:20 UTC-0500	MyLambdaFunction	DELETE_COMPLETE	-	-
2023-08-29 23:50:20 UTC-0500	MyApi	DELETE_IN_PROGRESS	-	-
2023-08-29 23:50:18 UTC-0500	HookTestStack	ROLLBACK_IN_PROGRESS	The following resource(s) failed to create: [MyLambdaFunction]. Rollback requested by user.	-
2023-08-29 23:50:17 UTC-0500	MyLambdaFunction	CREATE_FAILED	The following hook(s) failed: [AWSSamples::LambdaTracingCheck::Hook]	-
2023-08-29 23:50:17 UTC-0500	MyLambdaFunction	CREATE_IN_PROGRESS	-	AWSSamples::LambdaTracingCheck::Hook
2023-08-29 23:50:16 UTC-0500	MyLambdaFunction	CREATE_IN_PROGRESS	-	AWSSamples::LambdaTracingCheck::Hook
2023-08-29 23:50:15 UTC-0500	MyLambdaFunction	CREATE_IN_PROGRESS	-	-
2023-08-29 23:50:14 UTC-0500	LambdaExecutionRole	CREATE_COMPLETE	-	-
2023-08-29 23:49:59 UTC-0500	MyApi	CREATE_COMPLETE	-	-
2023-08-29 23:49:59 UTC-0500	MyApi	CREATE_IN_PROGRESS	Resource creation Initiated	-
2023-08-29 23:49:58 UTC-0500	LambdaExecutionRole	CREATE_IN_PROGRESS	Resource creation Initiated	-
2023-08-29 23:49:58 UTC-0500	LambdaExecutionRole	CREATE_IN_PROGRESS	-	-
2023-08-29 23:49:58 UTC-0500	MyApi	CREATE_IN_PROGRESS	-	-
2023-08-29 23:49:55 UTC-0500	HookTestStack	CREATE_IN_PROGRESS	User Initiated	-
2023-08-29 23:49:50 UTC-0500	HookTestStack	REVIEW_IN_PROGRESS	User Initiated	-

Jika Anda mengaktifkan log untuk CloudFormation hook Anda, Anda dapat menangkap hasil evaluasi kait. Berikut adalah contoh log untuk hook dengan status gagal, yang menunjukkan bahwa fungsi Lambda belum X-Ray diaktifkan:

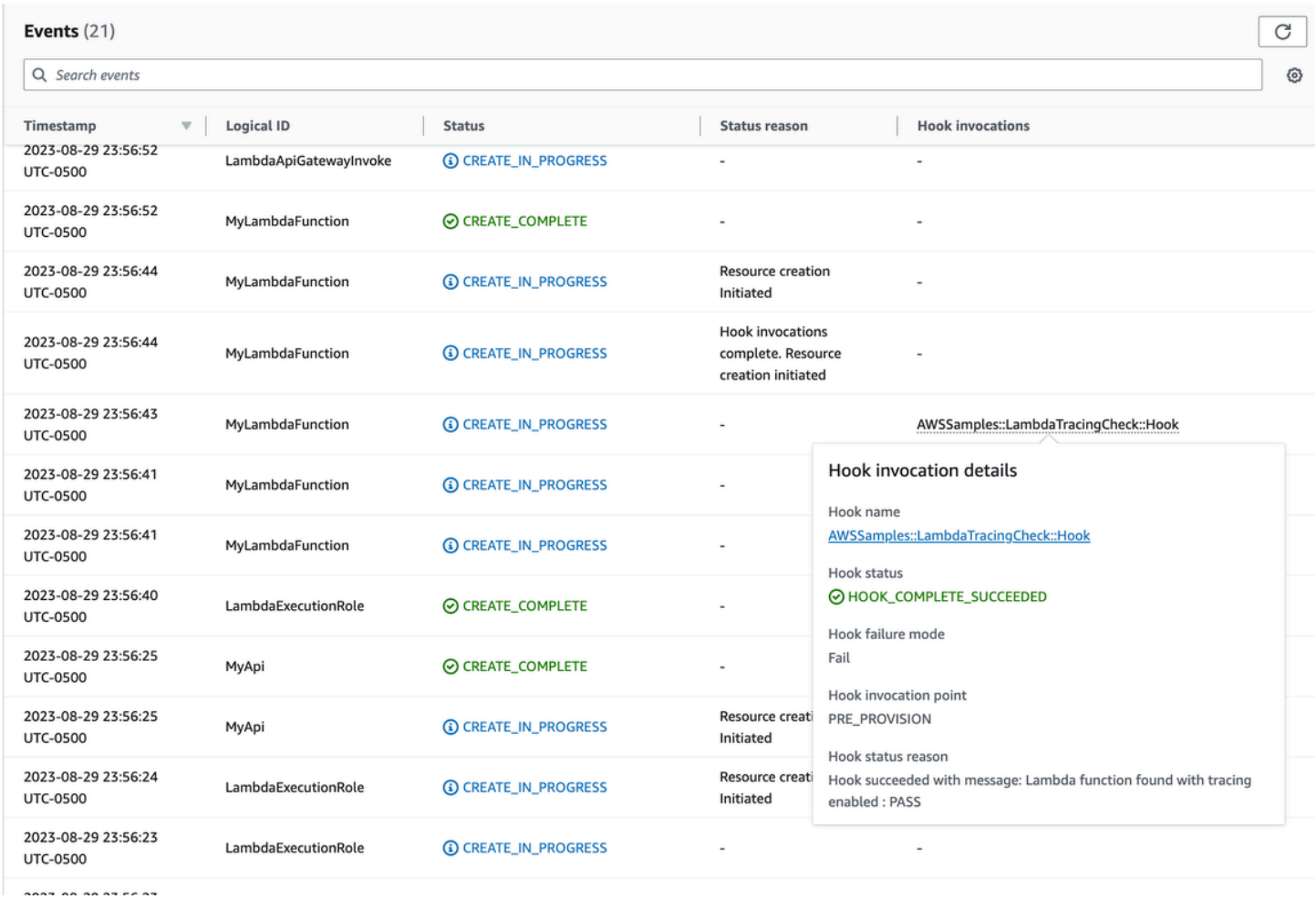
2023-08-29T23:50:17.574-05:00 ProgressEvent(status=<OperationStatus.FAILED: 'FAILED'>, errorCode=<HandlerErrorCode.NonCompliant: 'NonCompliant'…

```
ProgressEvent(status=<OperationStatus.FAILED: 'FAILED'>, errorCode=<HandlerErrorCode.NonCompliant: 'NonCompliant'>, message='Lambda
function found with no tracing enabled : FAILED', result=None, callbackContext=None, callbackDelaySeconds=0, resourceModel=None,
resourceModels=None, nextToken=None)
```

Copy

No newer events at this moment. *Auto retry paused.* Resume

Jika pengembang memilih untuk mengubah IAc untuk memperbarui `TracingConfig Mode` nilai ke `Active` dan menerapkan kembali, hook berhasil dijalankan dan tumpukan berlanjut dengan membuat sumber daya Lambda.



Dengan cara ini, Anda dapat menerapkan kontrol preventif dengan AWS Config dalam mode proaktif saat mengembangkan dan menerapkan sumber daya tanpa server di akun Anda. AWS Dengan mengintegrasikan AWS Config aturan ke dalam CI/CD pipeline, Anda dapat mengidentifikasi dan secara opsional memblokir penerapan sumber daya yang tidak sesuai, seperti fungsi Lambda yang tidak memiliki konfigurasi pelacakan aktif. Ini memastikan bahwa hanya sumber daya yang mematuhi kebijakan tata kelola terbaru yang digunakan ke AWS lingkungan Anda.

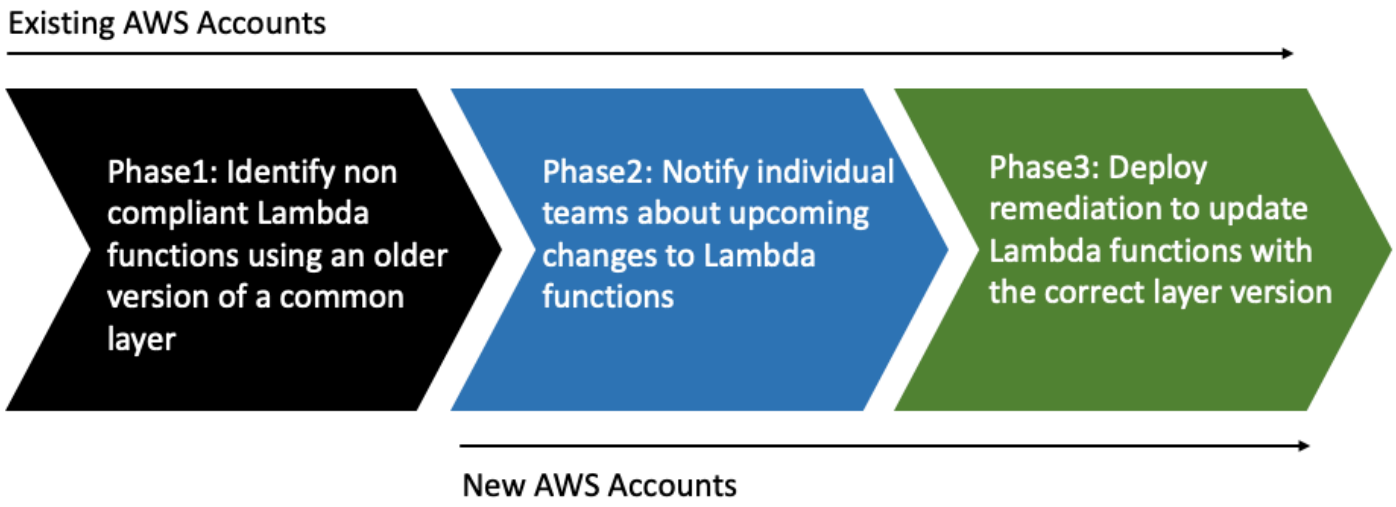
Mendeteksi penerapan dan konfigurasi Lambda yang tidak sesuai dengan AWS Config

Selain evaluasi pro aktif, Anda juga AWS Config dapat mendeteksi penerapan sumber daya dan konfigurasi secara reaktif yang tidak sesuai dengan kebijakan tata kelola Anda. Ini penting karena kebijakan tata kelola berkembang seiring organisasi Anda mempelajari dan menerapkan praktik terbaik baru.

Pertimbangkan skenario di mana Anda menetapkan kebijakan baru saat menerapkan atau memperbarui fungsi Lambda: Semua fungsi Lambda harus selalu menggunakan versi lapisan Lambda tertentu yang disetujui. Anda dapat mengonfigurasi AWS Config untuk memantau fungsi baru atau yang diperbarui untuk konfigurasi lapisan. Jika AWS Config mendeteksi fungsi yang tidak menggunakan versi lapisan yang disetujui, fungsi tersebut ditandai sebagai sumber daya yang tidak sesuai.

Anda dapat secara opsional AWS Config mengonfigurasi untuk memperbaiki sumber daya secara otomatis dengan menentukan tindakan remediasi menggunakan dokumen AWS Systems Manager otomatisasi. Misalnya, Anda dapat menulis dokumen otomatisasi dalam Python menggunakan AWS SDK untuk Python (Boto3), yang memperbarui fungsi yang tidak sesuai untuk menunjuk ke versi lapisan yang disetujui. Dengan demikian, AWS Config berfungsi sebagai detektif dan kontrol korektif, mengotomatiskan manajemen kepatuhan.

Mari kita uraikan proses ini menjadi tiga fase implementasi penting:



Tahap 1: Identifikasi sumber daya akses

Mulailah dengan mengaktifkan AWS Config di seluruh akun Anda dan mengonfigurasinya untuk merekam fungsi AWS Lambda. Hal ini memungkinkan AWS Config untuk mengamati kapan fungsi Lambda dibuat atau diperbarui. Anda kemudian dapat mengonfigurasi aturan kebijakan khusus untuk memeriksa pelanggaran kebijakan tertentu, yang menggunakan AWS CloudFormation Guard sintaks. Aturan penjaga mengambil bentuk umum berikut:

```
rule name when condition { assertion }
```

Di bawah ini adalah contoh aturan yang memeriksa untuk memastikan bahwa lapisan tidak disetel ke versi lapisan lama:

```
rule desiredlayer when configuration.layers !empty {
    some configuration.layers[*].arn != CONFIG_RULE_PARAMETERS.OldLayerArn
}
```

Mari kita pahami sintaks aturan dan struktur:

- Nama aturan: Nama aturan dalam contoh yang disediakan adalah`desiredlayer`.
- Kondisi: Klausul ini menentukan kondisi di mana aturan harus diperiksa. Dalam contoh yang diberikan, kondisinya adalah`configuration.layers !empty`. Ini berarti sumber daya harus dievaluasi hanya ketika `layers` properti dalam konfigurasi tidak kosong.
- Pernyataan: Setelah `when` klausa, pernyataan menentukan apa yang diperiksa aturan. Pernyataan `some configuration.layers[*].arn != CONFIG_RULE_PARAMETERS.OldLayerArn` memeriksa apakah salah satu ARN lapisan Lambda tidak cocok dengan nilainya. `OldLayerArn` Jika mereka tidak cocok, pernyataan itu benar dan aturan berlalu; jika tidak, itu gagal.

`CONFIG_RULE_PARAMETERS`adalah seperangkat parameter khusus yang dikonfigurasi dengan AWS Config aturan. Dalam hal ini, `OldLayerArn` adalah parameter di dalam`CONFIG_RULE_PARAMETERS`. Hal ini memungkinkan pengguna untuk memberikan nilai ARN tertentu yang mereka anggap lama atau usang, dan kemudian aturan memeriksa apakah ada fungsi Lambda yang menggunakan ARN lama ini.

Tahap 2: Visualisasikan dan desain

AWS Config mengumpulkan data konfigurasi dan menyimpan data tersebut di bucket Amazon Simple Storage Service (Amazon S3). Anda dapat menggunakan Amazon Athena untuk menanyakan data

ini langsung dari bucket S3 Anda. Dengan Athena, Anda dapat mengumpulkan data ini di tingkat organisasi, menghasilkan tampilan holistik konfigurasi sumber daya Anda di semua akun Anda. Untuk mengatur agregasi data konfigurasi sumber daya, lihat Mem visualisasikan AWS Config data menggunakan Athena dan Amazon Quick di blog AWS Cloud Operations and Management.

Berikut ini adalah contoh kueri Athena untuk mengidentifikasi semua fungsi Lambda menggunakan lapisan tertentu ARN:

```
WITH unnested AS (
    SELECT
      item.awsaccountid AS account_id,
      item.awsregion AS region,
      item.configuration AS lambda_configuration,
      item.resourceid AS resourceid,
      item.resourcename AS resourcename,
      item.configuration AS configuration,
      json_parse(item.configuration) AS lambda_json
    FROM
      default.aws_config_configuration_snapshot,
      UNNEST(configurationitems) as t(item)
    WHERE
      "dt" = 'latest'
      AND item.resourcetype = 'AWS::Lambda::Function'
  )

  SELECT DISTINCT
    region as Region,
    resourcename as FunctionName,
    json_extract_scalar(lambda_json, '$.memorySize') AS memory_size,
    json_extract_scalar(lambda_json, '$.timeout') AS timeout,
    json_extract_scalar(lambda_json, '$.version') AS version
  FROM
    unnested
  WHERE
    lambda_configuration LIKE '%arn:aws:lambda:us-
east-1:111122223333:layer:AnyGovernanceLayer:24%'
```

Berikut adalah hasil dari kueri:

Query results | Query stats

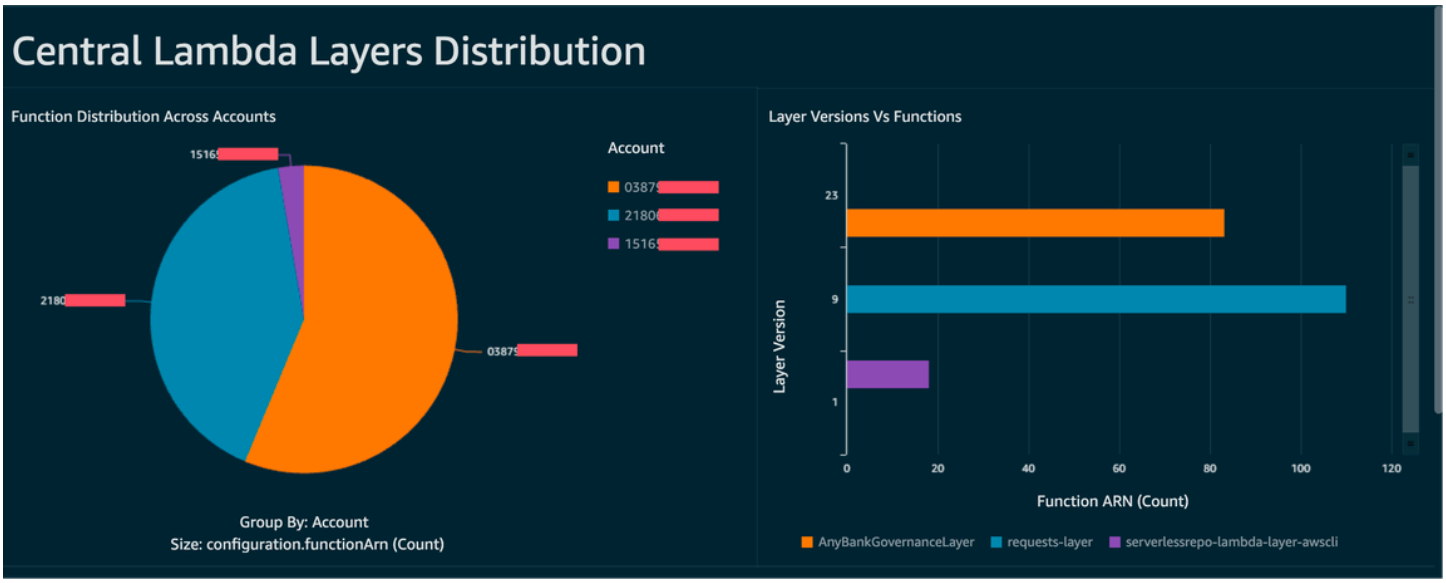
Completed — Time in queue: 127 ms — Run time: 1.803 sec — Data scanned: 239.40 KB

Results (27) — Copy — Download results

Search rows — 1

#	Region	FunctionName	memory_size	timeout	version
1	us-east-1	UpdateUIForPublishEvents	128	18	$LATEST
2	us-east-1	SchedulerCLI-InstanceSchedulerMain	128	300	$LATEST
3	us-east-1	my_functions_function10	128	3	$LATEST
4	us-east-1	lex-web-ui-CognitoIdentityP-CleanStackNameFunction-1TSORSH6L6YXQ	128	300	$LATEST
5	us-east-1	GetLatestArn	128	3	$LATEST
6	us-east-1	aws-python-http-api-project-dev-hello	1024	6	$LATEST
7	us-east-1	cloud9-MyTest-MyTest-688JGPVYP37L	128	15	$LATEST
8	us-east-1	my_functions_function1	128	3	$LATEST
9	us-east-1	my_functions_function25	128	3	$LATEST

Dengan AWS Config data yang dikumpulkan di seluruh organisasi, Anda kemudian dapat membuat dasbor menggunakan Amazon Quick. Dengan mengimpor hasil Athena Anda ke Quick, Anda dapat memvisualisasikan seberapa baik fungsi Lambda Anda mematuhi aturan versi lapisan. Dasbor ini dapat menyoroti sumber daya yang sesuai dan tidak sesuai, yang membantu Anda menentukan kebijakan penegakan hukum Anda, seperti yang diuraikan di bagian ??? berikutnya. Gambar berikut adalah contoh dasbor yang melaporkan distribusi versi lapisan yang diterapkan pada fungsi dalam organisasi.



Fase 3: Menerapkan dan menegakkan

Anda sekarang dapat secara opsional memasangkan aturan versi lapisan yang Anda buat di fase 1 dengan tindakan remediasi melalui dokumen otomatisasi Manajer Sistem, yang Anda buat sebagai skrip Python yang ditulis dengan AWS SDK untuk Python (Boto3). Skrip memanggil tindakan

UpdateFunctionConfiguration API untuk setiap fungsi Lambda, memperbarui konfigurasi fungsi dengan layer baru ARN. Atau, Anda dapat meminta skrip mengirimkan permintaan tarik ke repositori kode untuk memperbarui lapisan ARN. Dengan cara ini penerapan kode masa depan juga diperbarui dengan lapisan ARN yang benar.

Penandatanganan kode lambda dengan AWS Signer

AWS Signeradalah layanan penandatanganan kode yang dikelola sepenuhnya yang dapat Anda gunakan untuk memvalidasi kode Anda terhadap tanda tangan digital untuk mengonfirmasi bahwa kode tidak diubah dan dari penerbit tepercaya. AWS Signer dapat digunakan bersamaan dengan AWS Lambda untuk memverifikasi bahwa fungsi dan lapisan tidak berubah sebelum penerapan ke AWS lingkungan Anda. Ini melindungi organisasi Anda dari aktor jahat yang mungkin telah memperoleh kredensi untuk membuat fungsi baru atau memperbarui fungsi yang ada.

Untuk mengatur penandatanganan kode untuk fungsi Lambda Anda, mulailah dengan membuat bucket S3 dengan pembuatan versi diaktifkan. Setelah itu, buat profil penandatanganan dengan AWS Signer, tentukan Lambda sebagai platform dan kemudian tentukan periode hari di mana profil penandatanganan valid. Contoh:

```
Signer:
  Type: AWS::Signer::SigningProfile
  Properties:
    PlatformId: AWSLambda-SHA384-ECDSA
    SignatureValidityPeriod:
      Type: DAYS
      Value: !Ref pValidDays
```

Kemudian gunakan profil penandatanganan dan buat konfigurasi penandatanganan dengan Lambda. Anda harus menentukan apa yang harus dilakukan ketika konfigurasi penandatanganan melihat artefak yang tidak cocok dengan tanda tangan digital yang diharapkan: memperingatkan (tetapi izinkan penerapan) atau menegakkan (dan blokir penerapan). Contoh di bawah ini dikonfigurasi untuk menegakkan dan memblokir penerapan.

```
SigningConfig:
  Type: AWS::Lambda::CodeSigningConfig
  Properties:
    AllowedPublishers:
      SigningProfileVersionArns:
        - !GetAtt Signer.ProfileVersionArn
    CodeSigningPolicies:
      UntrustedArtifactOnDeployment: Enforce
```

Anda sekarang telah mengon AWS Signer figurasi dengan Lambda untuk memblokir penerapan yang tidak tepercaya. Mari kita asumsikan Anda telah selesai mengkodekan permintaan fitur dan

sekarang siap untuk menerapkan fungsi tersebut. Langkah pertama adalah memasang kode dengan dependensi yang sesuai dan kemudian menandatangani artefak menggunakan profil penandatanganan yang Anda buat. Anda dapat melakukan ini dengan mengunggah artefak zip ke bucket S3 dan kemudian memulai pekerjaan penandatanganan.

```
aws signer start-signing-job \
--source 's3={bucketName=your-versioned-bucket,key=your-prefix/your-zip-
artifact.zip,version=QyaJ3c4qa50LXV.9VaZgXHlsGbvCXxpT}' \
--destination 's3={bucketName=your-versioned-bucket,prefix=your-prefix/}' \
--profile-name your-signer-id
```

Anda mendapatkan output sebagai berikut, di mana `jobId` adalah objek yang dibuat di bucket tujuan dan awalan dan `jobOwner` merupakan Akun AWS ID 12 digit tempat pekerjaan dijalankan.

```
{
    "jobId": "87a3522b-5c0b-4d7d-b4e0-4255a8e05388",
    "jobOwner": "111122223333"
  }
```

Dan sekarang Anda dapat menerapkan fungsi Anda menggunakan objek S3 yang ditandatangani dan konfigurasi penandatanganan kode yang Anda buat.

```
  Fn:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: s3://your-versioned-bucket/your-prefix/87a3522b-5c0b-4d7d-
b4e0-4255a8e05388.zip
      Handler: fn.handler
      Role: !GetAtt FnRole.Arn
      CodeSigningConfigArn: !Ref pSigningConfigArn
```

Sebagai alternatif, Anda dapat menguji penyebaran fungsi dengan artefak zip sumber asli yang tidak ditandatangani. Penerapan akan gagal dengan pesan berikut:

```
Lambda cannot deploy the function. The function or layer might be signed using a
 signature that the client is not configured to accept. Check the provided signature
 for unsigned.
```

Jika Anda membangun dan menerapkan fungsi menggunakan AWS Serverless Application Model (AWS SAM), perintah package menangani pengunggahan artefak zip ke S3 dan juga memulai

pekerjaan penandatanganan dan mendapatkan artefak yang ditandatangani. Anda dapat melakukan ini dengan perintah dan parameter berikut:

```
sam package -t your-template.yaml \
--output-template-file your-output.yaml \
--s3-bucket your-versioned-bucket \
--s3-prefix your-prefix \
--signing-profiles your-signer-id
```

AWS Signer membantu Anda memverifikasi bahwa artefak zip yang digunakan ke akun Anda dipercaya untuk penerapan. Anda dapat menyertakan proses di atas dalam CI/CD pipeline Anda dan mengharuskan semua fungsi memiliki konfigurasi penandatanganan kode yang dilampirkan menggunakan teknik yang diuraikan dalam topik sebelumnya. Dengan menggunakan penandatanganan kode dengan penerapan fungsi Lambda, Anda mencegah aktor jahat yang mungkin mendapatkan kredenSIAL untuk membuat atau memperbarui fungsi menyuntikkan kode berbahaya ke fungsi Anda.

Otomatiskan penilaian keamanan untuk Lambda dengan Amazon Inspector

Amazon Inspector adalah layanan manajemen kerentanan yang terus memindai beban kerja untuk kerentanan perangkat lunak yang diketahui dan paparan jaringan yang tidak diinginkan. Amazon Inspector membuat temuan yang menjelaskan kerentanan, mengidentifikasi sumber daya yang terpengaruh, menilai tingkat keparahan kerentanan, dan memberikan panduan remediasi.

Dukungan Amazon Inspector menyediakan penilaian kerentanan keamanan otomatis yang berkelanjutan untuk fungsi dan lapisan Lambda. Amazon Inspector menyediakan dua jenis pemindaian untuk Lambda:

- Pemindaian standar Lambda (default): Mem indai dependensi aplikasi dalam fungsi Lambda dan lapisannya untuk kerentanan paket.
- Pemindaian kode lambda: Memindai kode aplikasi khusus di fungsi dan lapisan Anda untuk kerentanan https://docs.aws.amazon.com/inspector/latest/user/findings-types.html#findings-types-code kode. Anda dapat mengaktifkan pemindaian standar Lambda atau mengaktifkan pemindaian standar Lambda bersama dengan pemindaian kode Lambda.

Untuk mengaktifkan Amazon Inspector, navigasikan ke konsol Amazon Inspector, per luas bagian Pengaturan, dan pilih Manajemen Akun. Pada tab Akun, pilih Aktif kan, lalu pilih salah satu opsi pemindaian.

Anda dapat mengaktifkan Amazon Inspector untuk beberapa akun dan mendelegasikan izin untuk mengelola Amazon Inspector untuk organisasi ke akun tertentu saat menyiapkan Amazon Inspector. Saat mengaktifkan, Anda perlu memberikan izin Amazon Inspector dengan membuat peran:`AWSServiceRoleForAmazonInspector2`. Di konsol Amazon Inspector, Anda dapat membuat peran ini menggunakan opsi satu langkah.

Untuk pemindaian standar Lambda, Amazon Inspector memulai pemindaian kerentanan fungsi Lambda dalam situasi berikut:

- Segera setelah Amazon Inspector menemukan fungsi Lambda yang ada.
- Saat Anda menerapkan fungsi Lambda baru.
- Saat Anda menerapkan pembaruan ke kode aplikasi atau dependensi dari fungsi Lambda yang ada atau lapisannya.
- Setiap kali Amazon Inspector menambahkan item kerentanan dan eksposur umum (CVE) baru ke database-nya, dan CVE tersebut relevan dengan fungsi Anda.

Untuk pemindaian kode Lambda, Amazon Inspector mengevaluasi kode aplikasi fungsi Lambda menggunakan penalaran otomatis dan pembelajaran mesin yang menganalisis kode aplikasi Anda untuk kepatuhan keamanan secara keseluruhan. Jika Amazon Inspector mendeteksi kerentanan dalam kode aplikasi fungsi Lambda Anda, Amazon Inspector menghasilkan temuan Kerentanan Kode yang terperinci. Untuk daftar kemungkinan deteksi, lihat Perpustakaan Det CodeGuru ektor Amazon.

Untuk melihat temuan, buka konsol Amazon Inspector. Pada menu F indings, pilih By Lambda function untuk menampilkan hasil pemindaian keamanan yang dilakukan pada fungsi Lambda.

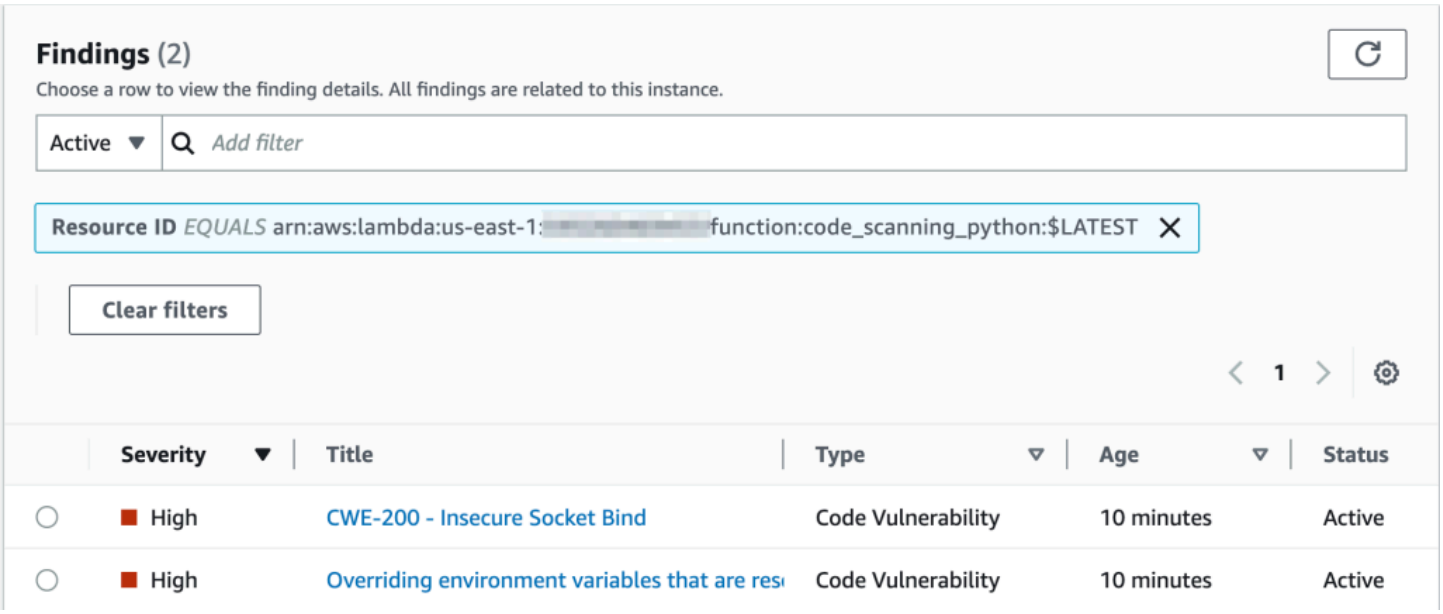
Untuk mengecualikan fungsi Lambda dari pemindaian standar, beri tag fungsi dengan pasangan nilai kunci berikut:

- `Key:InspectorExclusion`
- `Value:LambdaStandardScanning`

Untuk mengecualikan fungsi Lambda dari pemindaian kode, beri tag fungsi dengan pasangan kunci-nilai berikut:

- `Key:InspectorCodeExclusion`
- `Value:LambdaCodeScanning`

Misalnya, seperti yang ditunjukkan pada gambar berikut, Amazon Inspector secara otomatis mendeteksi kerentanan dan mengkategorikan temuan jenis Kerentanan Kode, yang menunjukkan bahwa kerentanan ada dalam kode fungsi, dan bukan di salah satu pustaka yang bergantung pada kode. Anda dapat memeriksa detail ini untuk fungsi tertentu atau beberapa fungsi sekaligus.



Anda dapat menyelami lebih jauh ke dalam setiap temuan ini dan mempelajari cara mengatasi masalah tersebut.

Overriding environment variables that are reserved by AWS Lambda might lead to unexpected behavior.

Finding ID: arn:aws:inspector2:us-east-1:[redacted]:finding/[redacted]

Overriding environment variables that are reserved by AWS Lambda might lead to unexpected behavior or failure of the Lambda function.

Finding overview	
AWS account ID	[redacted]
Severity	High
Type	Code Vulnerability
Detector name	Override of reserved variable names in a Lambda function
Relevant CWE	--
Rule ID	Rule-434311
Detector tags	#availability, #aws-python-sdk, #aws-lambda, #data-integrity, #maintainability, #security, #security-context, #python
Fix available	Yes
Created at	March 29, 2023 10:08 AM (UTC-04:00)

Vulnerability details	
File path	lambda_function.py

Vulnerability location

```
3  import socket
4
5  def lambda_handler(event, context):
6
7      # print("Scenario 1");
8      os.environ['_HANDLER'] = 'hello'
9      # print("Scenario 1 ends")
10
11     # print("Scenario 2");
12     s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
13     s.bind(('',0))
```

Suggested remediation

Your code attempts to override an environment variable that is reserved by the Lambda runtime environment. This can lead to unexpected behavior and might break the execution of your Lambda function.

Saat bekerja dengan fungsi Lambda Anda, pastikan Anda mematuhi konvensi penamaan untuk fungsi Lambda Anda. Untuk informasi selengkapnya, lihat Bekerja dengan variabel lingkungan Lambda.

Anda bertanggung jawab atas saran remediasi yang Anda terima. Selalu tinjau saran remediasi sebelum menerimanya. Anda mungkin perlu mengedit saran remediasi untuk memastikan bahwa kode Anda melakukan apa yang Anda inginkan.

Menerapkan observabilitas untuk keamanan dan kepatuhan Lambda

AWS Config adalah alat yang berguna untuk menemukan dan memperbaiki sumber daya AWS Serverless yang tidak sesuai. Setiap perubahan yang Anda buat pada sumber daya tanpa server Anda dicatat di AWS Config. Selain itu, dengan AWS Config, Anda dapat menyimpan data snapshot konfigurasi di S3.

Anda dapat menggunakan Amazon Athena dan Amazon Quick untuk membuat dasbor dan melihat AWS Config data. DiMendeteksi penerapan dan konfigurasi Lambda yang tidak sesuai dengan AWS Config, kami membahas bagaimana kami dapat memvisualisasikan konfigurasi tertentu seperti lapisan Lambda. Topik ini memperluas konsep-konsep ini.

Visibilitas ke konfigurasi Lambda

Anda dapat menggunakan kueri untuk menarik konfigurasi penting seperti Akun AWS ID, Wilayah, konfigurasi AWS X-Ray pelacakan, konfigurasi VPC, ukuran memori, runtime, dan tag. Berikut adalah contoh kueri yang dapat Anda gunakan untuk menarik informasi ini dari Athena:

```
WITH unnested AS (
    SELECT
      item.awsaccountid AS account_id,
      item.awsregion AS region,
      item.configuration AS lambda_configuration,
      item.resourceid AS resourceid,
      item.resourcename AS resourcename,
      item.configuration AS configuration,
      json_parse(item.configuration) AS lambda_json
    FROM
      default.aws_config_configuration_snapshot,
      UNNEST(configurationitems) as t(item)
    WHERE
      "dt" = 'latest'
      AND item.resourcetype = 'AWS::Lambda::Function'
  )

  SELECT DISTINCT
    account_id,
    tags,
    region as Region,
    resourcename as FunctionName,
    json_extract_scalar(lambda_json, '$.memorySize') AS memory_size,
```

```
    json_extract_scalar(lambda_json, '$.timeout') AS timeout,
    json_extract_scalar(lambda_json, '$.runtime') AS version
    json_extract_scalar(lambda_json, '$.vpcConfig.SubnetIds') AS vpcConfig
    json_extract_scalar(lambda_json, '$.tracingConfig.mode') AS tracingConfig
  FROM
    unnested
```

Anda dapat menggunakan kueri untuk membangun dasbor Cepat dan memvisualisasikan data. Untuk mengumpulkan data konfigurasi AWS sumber daya, membuat tabel di Athena, dan membangun dasbor Cepat pada data dari Athena, lihat Mem visualisasikan AWS Config data menggunakan Athena dan Amazon Quick di blog AWS Cloud Operations and Management. Khususnya, kueri ini juga mengambil informasi tag untuk fungsi. Hal ini memungkinkan wawasan yang lebih dalam tentang beban kerja dan lingkungan Anda, terutama jika Anda menggunakan tag khusus.

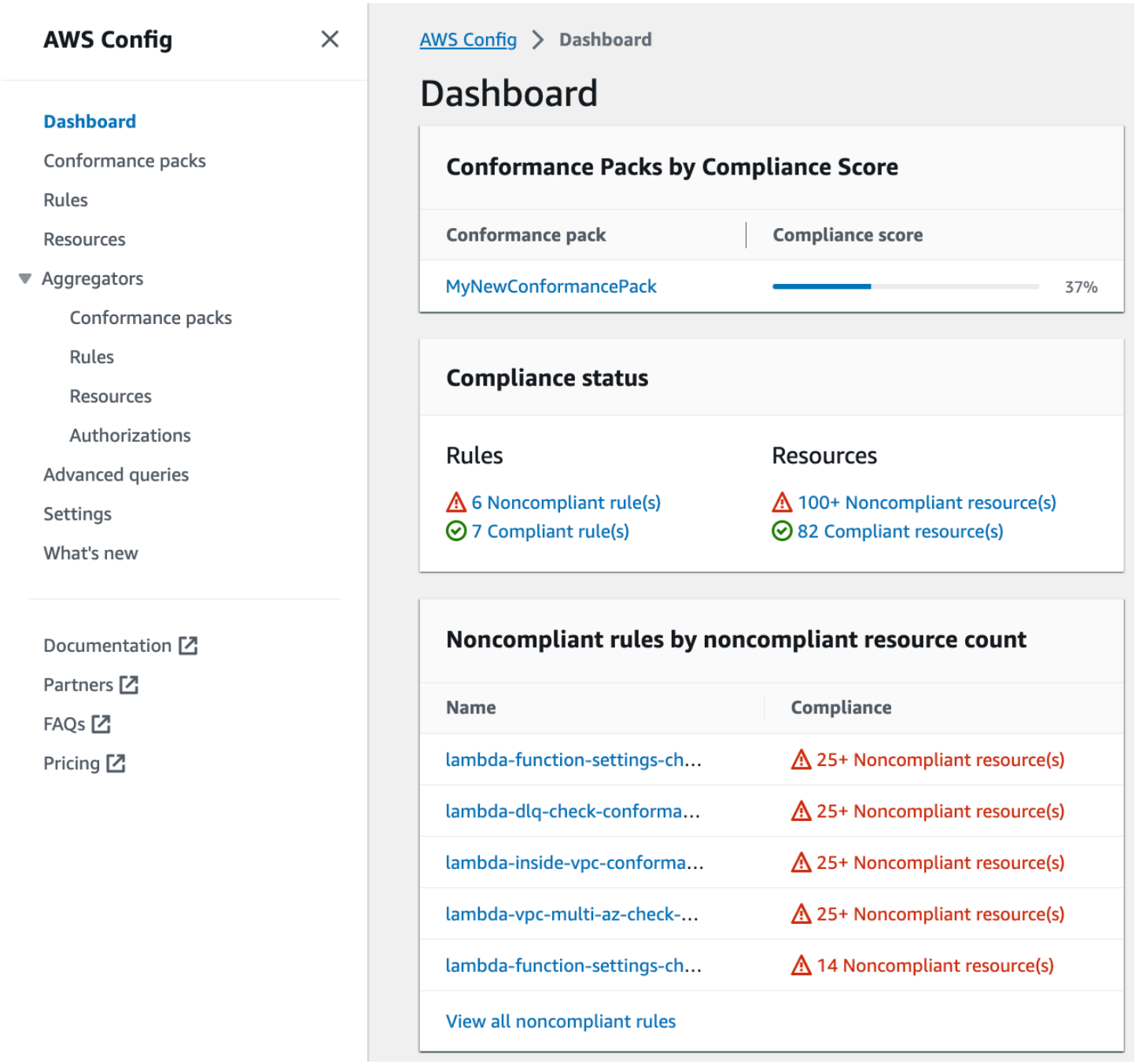


Untuk informasi selengkapnya tentang tindakan yang dapat Anda lakukan, lihat Mengatasi temuan observabilitas bagian selanjutnya dalam topik ini.

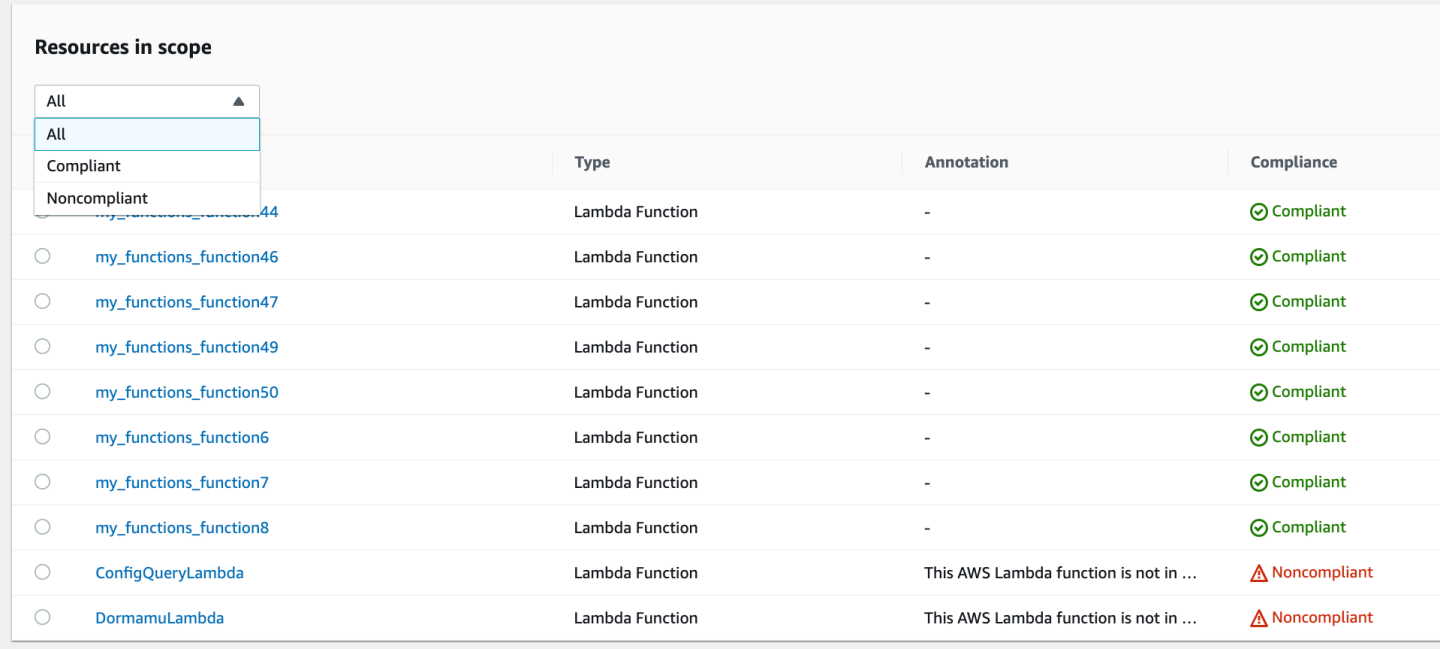
Visibilitas ke kepatuhan Lambda

Dengan data yang dihasilkan oleh AWS Config, Anda dapat membuat dasbor tingkat organisasi untuk memantau kepatuhan. Hal ini memungkinkan pelacakan dan pemantauan yang konsisten terhadap:

- Paket kepatuhan berdasarkan skor kepatuhan
- Aturan oleh sumber daya yang tidak sesuai
- Status kepatuhan

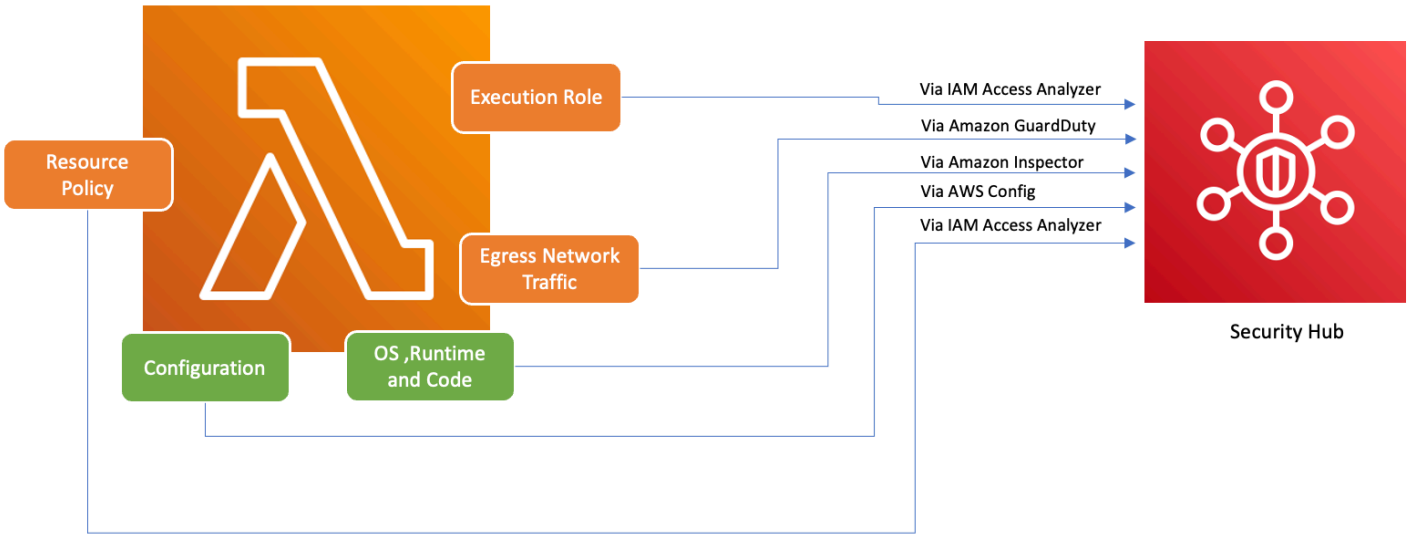


Periksa setiap aturan untuk mengidentifikasi sumber daya yang tidak sesuai untuk aturan tersebut. Misalnya, jika organisasi Anda mengamanatkan bahwa semua fungsi Lambda harus dikaitkan dengan VPC dan jika Anda telah menerapkan AWS Config aturan untuk mengidentifikasi kepatuhan, Anda dapat memilih `lambda-inside-vpc` aturan dalam daftar di atas.



Untuk informasi selengkapnya tentang tindakan yang dapat Anda lakukan, lihat Mengatasi temuan observabilitas bagian di bawah ini.

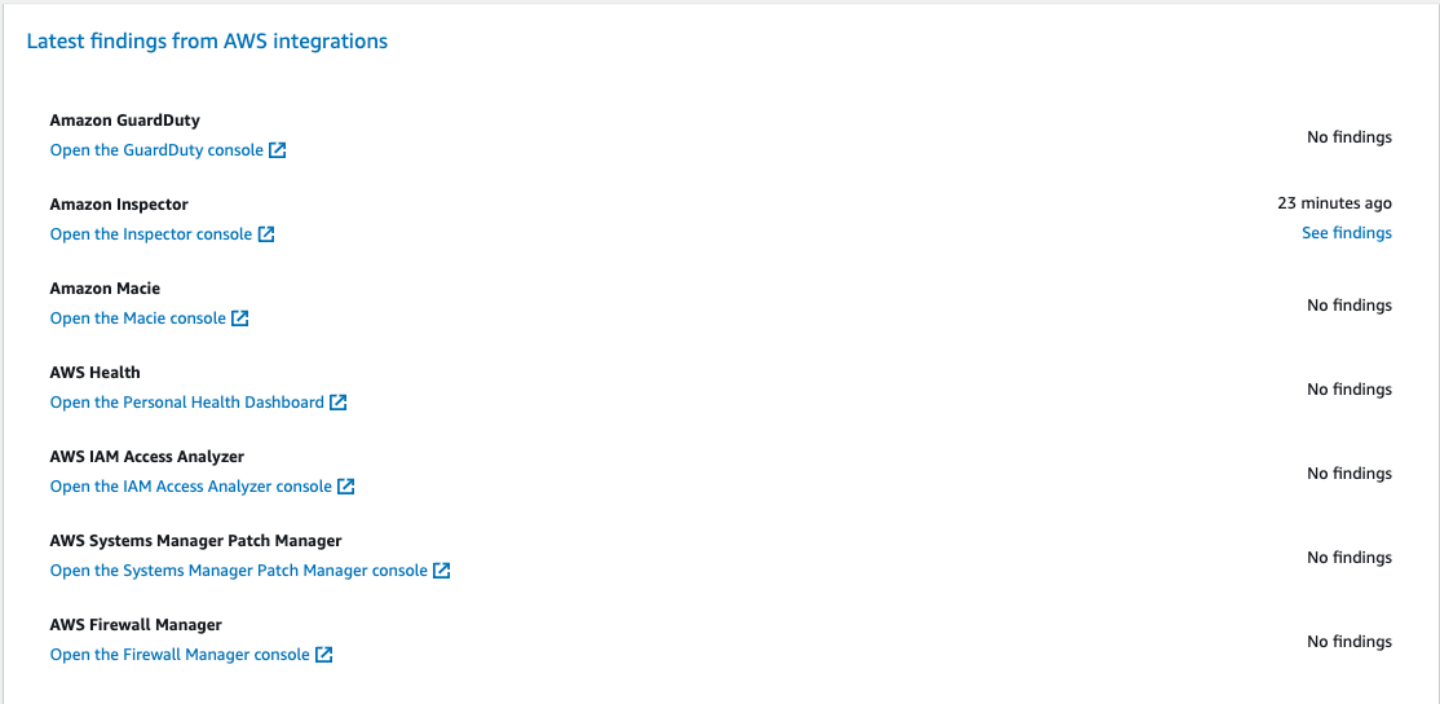
Visibilitas ke batas fungsi Lambda menggunakan Security Hub CSPM



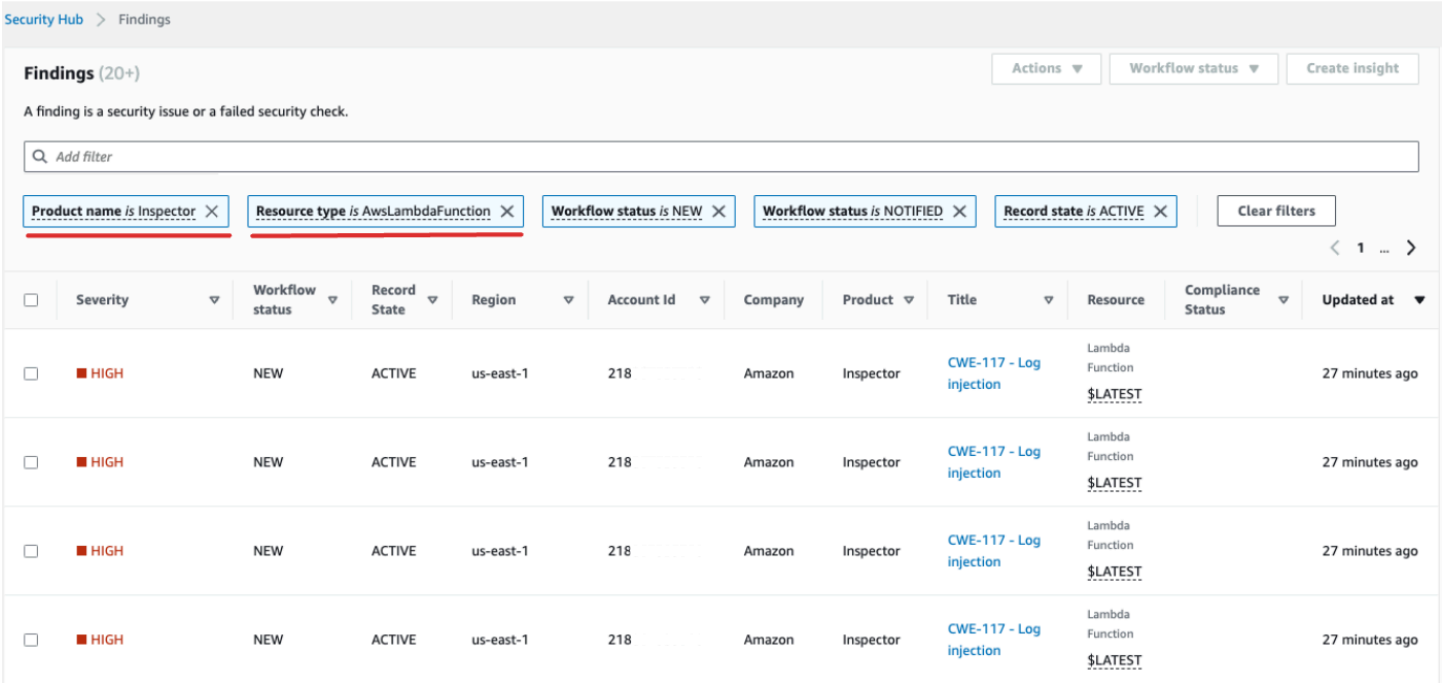
Untuk memastikan bahwa AWS layanan termasuk Lambda digunakan dengan aman, AWS memperkenalkan Praktik Terbaik Keamanan Dasar v1.0.0. Serangkaian praktik terbaik ini memberikan pedoman yang jelas untuk mengamankan sumber daya dan data di AWS lingkungan, menekankan pentingnya mempertahankan postur keamanan yang kuat. Ini AWS Security Hub CSPM melengkapi ini dengan menawarkan pusat keamanan dan kepatuhan terpadu. Ini mengumpulkan,

mengatur, dan memprioritaskan temuan keamanan dari beberapa AWS layanan seperti Amazon Inspector, AWS Identity and Access Management Access Analyzer, dan Amazon. GuardDuty

Jika Anda memiliki Security Hub CSPM, Amazon Inspector, IAM Access Analyzer, dan GuardDuty diaktifkan dalam AWS organisasi Anda, Security Hub CSPM secara otomatis mengumpulkan temuan dari layanan ini. Misalnya, mari kita lihat Amazon Inspector. Menggunakan Security Hub CSPM, Anda dapat secara efisien mengidentifikasi kode dan kerentanan paket dalam fungsi Lambda. Di konsol CSPM Security Hub, navigasikan ke bagian bawah berlabel Temuan terbaru dari AWS integrasi. Di sini, Anda dapat melihat dan menganalisis temuan yang bersumber dari berbagai AWS layanan terintegrasi.



Untuk melihat detail, pilih tautan Lihat temuan di kolom kedua. Ini menampilkan daftar temuan yang difilter berdasarkan produk, seperti Amazon Inspector. Untuk membatasi pencarian Anda ke fungsi Lambda, setel `ResourceType` ke`AwsLambdaFunction`. Ini menampilkan temuan dari Amazon Inspector terkait dengan fungsi Lambda.



Untuk GuardDuty, Anda dapat mengidentifikasi pola lalu lintas jaringan yang mencurigakan. Anomali semacam itu mungkin menunjukkan adanya kode yang berpotensi berbahaya dalam fungsi Lambda Anda.

Dengan IAM Access Analyzer, Anda dapat memeriksa kebijakan, terutama yang memiliki pernyataan kondisi yang memberikan akses fungsi ke entitas eksternal. Selain itu, IAM Access Analyzer mengevaluasi izin yang ditetapkan saat menggunakan AddPermission operasi di Lambda API bersama. `EventSourceToken`

Mengatasi temuan observabilitas

Mengingat konfigurasi luas yang mungkin untuk fungsi Lambda dan persyaratannya yang berbeda, solusi otomatisasi standar untuk remediasi mungkin tidak sesuai dengan setiap situasi. Selain itu, perubahan diterapkan secara berbeda di berbagai lingkungan. Jika Anda menemukan konfigurasi yang tampaknya tidak sesuai, pertimbangkan panduan berikut:

1. Strategi penandaan

   Kami merekomendasikan menerapkan strategi penandaan yang komprehensif. Setiap fungsi Lambda harus ditandai dengan informasi kunci seperti:

   - Pemilik: Orang atau tim yang bertanggung jawab atas fungsi tersebut.
   - Lingkungan: Produksi, pementasan, pengembangan, atau kotak pasir.

- Aplikasi: Konteks yang lebih luas dari fungsi ini, jika berlaku.

2. Penjangkauan pemilik

   Alih-alih mengotomatiskan perubahan yang rusak (seperti penyesuaian konfigurasi VPC), secara proaktif hubungi pemilik fungsi yang tidak sesuai (diidentifikasi oleh tag pemilik) dan berikan mereka waktu yang cukup untuk:

   - Sesuaikan konfigurasi yang tidak sesuai pada fungsi Lambda.
   - Berikan penjelasan dan minta pengecualian, atau perbaiki standar kepatuhan.

3. Memelihara database manajemen konfigurasi (CMDB)

   Sementara tag dapat memberikan konteks langsung, mempertahankan CMDB terpusat dapat memberikan wawasan yang lebih dalam. Ini dapat menyimpan informasi yang lebih terperinci tentang setiap fungsi Lambda, dependensinya, dan metadata penting lainnya. CMDB adalah sumber daya yang tak ternilai untuk audit, pemeriksaan kepatuhan, dan mengidentifikasi pemilik fungsi.

Karena lanskap infrastruktur tanpa server terus berkembang, penting untuk mengadopsi sikap proaktif terhadap pemantauan. Dengan alat seperti AWS Config, Security Hub CSPM, dan Amazon Inspector, potensi anomali atau konfigurasi yang tidak sesuai dapat diidentifikasi dengan cepat. Namun, alat saja tidak dapat memastikan kepatuhan total atau konfigurasi optimal. Sangat penting untuk memasangkan alat-alat ini dengan proses yang terdokumentasi dengan baik dan praktik terbaik.

- Umpan balik: Setelah langkah-langkah remediasi dilakukan, pastikan ada loop umpan balik. Ini berarti meninjau kembali sumber daya yang tidak sesuai secara berkala untuk mengonfirmasi apakah sumber daya tersebut telah diperbarui atau masih berjalan dengan masalah yang sama.
- Dokumentasi: Selalu dokumentasikan pengamatan, tindakan yang diambil, dan pengecualian yang diberikan. Dokumentasi yang tepat tidak hanya membantu selama audit tetapi juga membantu dalam meningkatkan proses untuk kepatuhan dan keamanan yang lebih baik di masa depan.
- Pelatihan dan kesadaran: Mem astikan bahwa semua pemangku kepentingan, terutama pemilik fungsi Lambda, dilatih secara teratur dan disadarkan akan praktik terbaik, kebijakan organisasi, dan mandat kepatuhan. Lokakarya reguler, webinar, atau sesi pelatihan dapat sangat membantu dalam memastikan semua orang berada di halaman yang sama dalam hal keamanan dan kepatuhan.

Sebagai kesimpulan, sementara alat dan teknologi memberikan kemampuan yang kuat untuk mendeteksi dan menandai potensi masalah, elemen manusia—pemahaman, komunikasi, pelatihan, dan dokumentasi—tetap sangat penting. Bersama-sama, mereka membentuk kombinasi yang kuat untuk memastikan bahwa fungsi Lambda Anda dan infrastruktur yang lebih luas tetap sesuai, aman, dan dioptimalkan untuk kebutuhan bisnis Anda.

Validasi kepatuhan untuk AWS Lambda

Third-party auditor menilai keamanan dan kepatuhan AWS Lambda sebagai bagian dari beberapa program AWS kepatuhan. Program ini mencakup SOC, PCI, FedRAMP, HIPAA, dan lainnya.

Untuk daftar AWS layanan dalam lingkup program kepatuhan tertentu, lihat AWS layanan dalam lingkup berdasarkan program kepatuhan. Untuk informasi umum, lihat Program kepatuhan AWS.

Anda dapat mengunduh laporan audit pihak ketiga menggunakan AWS Artifact. Untuk informasi lebih lanjut, lihat Mengunduh Laporan di AWS Artifact.

Tanggung jawab kepatuhan Anda saat menggunakan Lambda ditentukan oleh sensitivitas data Anda, tujuan kepatuhan perusahaan Anda, serta undang-undang dan peraturan yang berlaku. Anda dapat menerapkan kontrol tata kelola untuk memastikan bahwa fungsi Lambda perusahaan Anda memenuhi persyaratan kepatuhan Anda. Untuk informasi selengkapnya, lihat Buat strategi tata kelola untuk fungsi dan lapisan Lambda.

Untuk informasi tentang cara AWS melindungi infrastruktur global yang menjalankan Lambda, lihatthe section called “Keamanan infrastruktur”.

Ketahanan di AWS Lambda

Infrastruktur AWS global dibangun di sekitar Wil AWS ayah dan Zona Ketersediaan. AWS Wilayah menyediakan beberapa Zona Ketersediaan yang terpisah secara fisik dan terisolasi, yang terhubung dengan latensi rendah, throughput tinggi, dan jaringan yang sangat redundan. Dengan Zona Ketersediaan, Anda dapat merancang dan mengoperasikan aplikasi dan basis data yang secara otomatis melakukan failover di antara Zona Ketersediaan tanpa gangguan. Zona Ketersediaan memiliki ketersediaan dan toleransi kesalahan yang lebih baik, dan dapat diskalakan dibandingkan infrastruktur biasa yang terdiri dari satu atau beberapa pusat data.

Untuk informasi selengkapnya tentang AWS Wilayah dan Zona Ketersediaan, lihat infrastruktur AWS global.

Selain infrastruktur AWS global, Lambda menawarkan beberapa fitur untuk membantu mendukung ketahanan data dan kebutuhan cadangan Anda.

- Versioning – Anda dapat menggunakan versioning di Lambda untuk menyimpan kode dan konfigurasi fungsi saat Anda mengembangkannya. Bersama dengan alias, Anda dapat menggunakan pembuatan versi untuk melakukan blue/green dan melakukan penerapan bergulir. Untuk detailnya, lihat Kelola versi Fungsi Acara Lambda.

- Penskalaan – Ketika fungsi Anda menerima permintaan saat sedang memproses permintaan sebelumnya, Lambda meluncurkan instans lain dari fungsi Anda untuk menangani peningkatan beban. Lambda secara otomatis menyesuaikan skala untuk menangani 1.000 eksekusi bersamaan per Wilayah, dan kuota ini dapat ditingkatkan jika diperlukan. Untuk detailnya, lihat Memahami penskalaan Fungsi Acara Lambda.
- Ketersediaan tinggi – Lambda menjalankan fungsi Anda di beberapa Availability Zone guna memastikannya tersedia untuk memproses kejadian jika terjadi gangguan layanan dalam satu zona. Jika Anda mengonfigurasikan fungsi Anda agar terhubung ke virtual private cloud (VPC) di akun Anda, tentukan subnet di beberapa Availability Zone untuk memastikan ketersediaan yang tinggi. Untuk detailnya, lihat Memberikan Lambda Event Functions akses ke sumber daya di Amazon VPC.
- Konkurensi cadangan – Untuk memastikan bahwa fungsi Anda dapat selalu menskalakan untuk menangani permintaan tambahan, Anda dapat mencadangkan konkurensi untuk hal itu. Menetapkan konkurensi cadangan untuk suatu fungsi memastikan bahwa fungsi dapat menskalakan ke, tetapi tidak melebihi, jumlah invokasi bersamaan yang ditentukan. Hal ini memastikan bahwa Anda tidak kehilangan permintaan karena fungsi lain menggunakan semua konkurensi yang tersedia. Untuk detailnya, lihat Mengkonfigurasi konkurensi cadangan untuk Fungsi Acara.
- Percobaan ulang – Untuk invokasi asinkron dan subset invokasi yang dipicu oleh layanan lain, Lambda secara otomatis melakukan percobaan ulang terkait kesalahan dengan penundaan di antara percobaan. Klien lain dan Layanan AWS yang memanggil fungsi secara serempak bertanggung jawab untuk melakukan percobaan ulang. Lihat perinciannya di Memahami perilaku coba lagi di Fungsi Peristiwa Lambda.
- Dead-letter antrian — Untuk pemanggilan asinkron, Anda dapat mengonfigurasi Lambda untuk mengirim permintaan ke antrian huruf mati jika semua percobaan ulang gagal. Antrean surat mati adalah topik Amazon SNS atau antrean Amazon SQS yang menerima kejadian untuk pemecahan masalah atau pemrosesan ulang. Untuk detailnya, lihat Menambahkan antrian huruf mati.

Untuk informasi selengkapnya tentang jaringan dan infrastruktur fisik yang melindungi Lambda, lihatthe section called “Keamanan infrastruktur”.

Keamanan infrastruktur di AWS Lambda

Sebagai layanan terkelola, AWS Lambda dilindungi oleh keamanan jaringan AWS global. Untuk informasi tentang layanan AWS keamanan dan cara AWS melindungi infrastruktur, lihat Keamanan

AWS Cloud. Untuk mendesain AWS lingkungan Anda menggunakan praktik terbaik untuk keamanan infrastruktur, lihat Perlindungan Infrastruktur dalam Kerangka Kerja yang AWS Diarsiteksikan dengan Baik Pilar Keamanan.

Anda menggunakan panggilan API yang AWS dipublikasikan untuk mengakses Lambda melalui jaringan. Klien harus mendukung hal-hal berikut:

- Keamanan Lapisan Pengangkutan (TLS). Kami mensyaratkan TLS 1.2 dan menganjurkan TLS 1.3.
- Sandi cocok dengan sistem kerahasiaan maju sempurna (perfect forward secrecy, PFS) seperti DHE (Ephemeral Diffie-Hellman) atau ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Sebagian besar sistem modern seperti Java 7 dan versi lebih baru mendukung mode-mode ini.

Mengamankan beban kerja dengan titik akhir publik

Untuk beban kerja yang dapat diakses secara publik, AWS menyediakan sejumlah fitur dan layanan yang dapat membantu mengurangi risiko tertentu. Bagian ini mencakup otentikasi dan otorisasi pengguna aplikasi dan melindungi titik akhir API.

Autentikasi dan otorisasi

Otentikasi berkaitan dengan identitas dan otorisasi mengacu pada tindakan. Gunakan otentikasi untuk mengontrol siapa yang dapat memanggil fungsi Lambda, dan kemudian gunakan otorisasi untuk mengontrol apa yang dapat mereka lakukan. Untuk banyak aplikasi, IAM cukup untuk mengelola kedua mekanisme kontrol.

Untuk aplikasi dengan pengguna eksternal, seperti aplikasi web atau seluler, adalah umum untuk menggunakan Token Web JSON (JWT) untuk mengelola otentikasi dan otorisasi. Tidak seperti manajemen kata sandi berbasis server tradisional, JWT diteruskan dari klien pada setiap permintaan. Mereka adalah cara yang aman secara kriptografis untuk memverifikasi identitas dan klaim menggunakan data yang diteruskan dari klien. Untuk Lambda-based aplikasi, Anda dapat mengamankan setiap panggilan ke setiap titik akhir API tanpa mengandalkan server pusat untuk otentikasi.

Anda dapat menerapkan JWT dengan Amazon Cognito, layanan direktori pengguna yang dapat menangani pendaftaran, otentikasi, pemulihan akun, dan operasi manajemen akun umum lainnya. Amplify Framework menyediakan pustaka untuk menyederhanakan integrasi layanan ini ke dalam aplikasi frontend Anda. Anda juga dapat mempertimbangkan layanan mitra pihak ketiga seperti Auth0.

Mengingat peran keamanan penting dari layanan penyedia identitas, penting untuk menggunakan perkakas profesional untuk melindungi aplikasi Anda. Anda tidak disarankan untuk menulis layanan Anda sendiri untuk menangani otentikasi atau otorisasi. Kerentanan apa pun di pustaka khusus mungkin memiliki implikasi signifikan terhadap keamanan beban kerja Anda dan datanya.

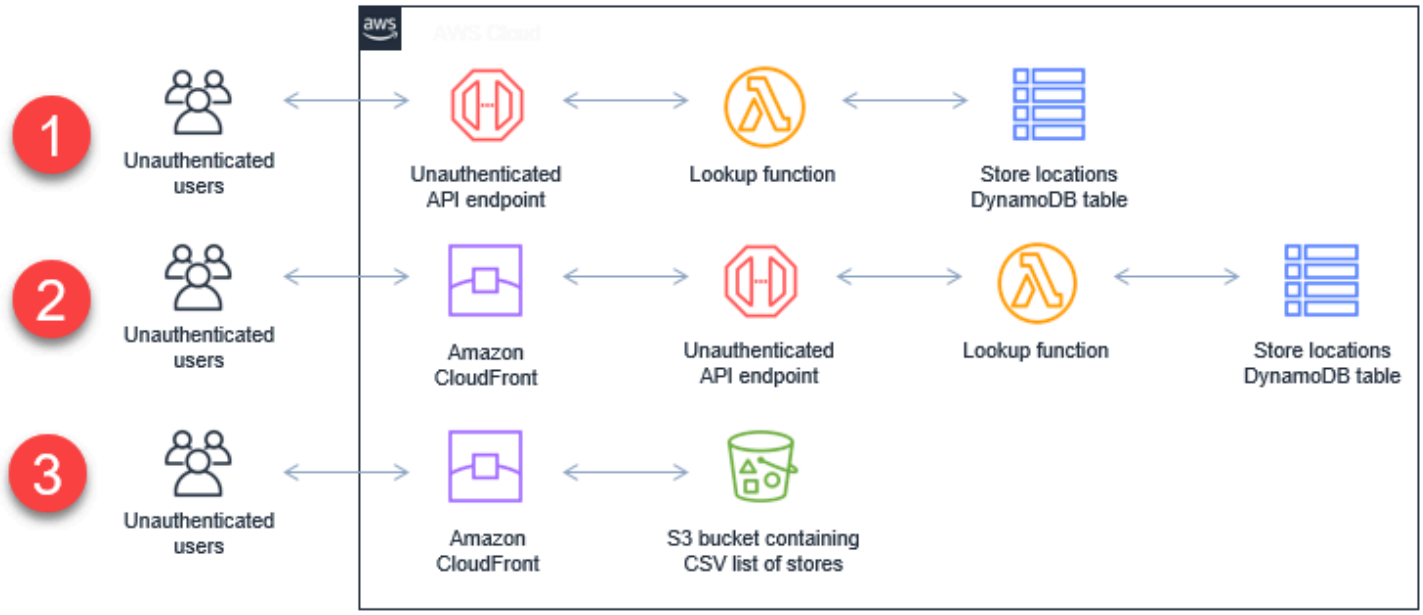
Melindungi titik akhir API

Untuk aplikasi tanpa server, cara yang lebih disukai untuk melayani aplikasi backend secara publik adalah dengan menggunakan Amazon API Gateway. Ini dapat membantu Anda melindungi API dari pengguna jahat atau lonjakan lalu lintas.

API Gateway menawarkan dua tipe titik akhir untuk pengembang tanpa server: REST API dan HTTP API. Keduanya mendukung otorisasi menggunakan AWS Lambda, IAM, atau Amazon Cognito. Saat menggunakan IAM atau Amazon Cognito, permintaan masuk dievaluasi dan jika mereka kehilangan token yang diperlukan atau berisi otentikasi yang tidak valid, permintaan ditolak. Anda tidak dikenakan biaya untuk permintaan ini dan mereka tidak diperhitungkan dalam kuota pembatasan apa pun.

Rute API yang tidak diautentikasi dapat diakses oleh siapa saja di internet publik sehingga disarankan agar Anda membatasi penggunaan API yang tidak diautentikasi. Jika Anda harus menggunakan API yang tidak diautentikasi, penting untuk melindunginya dari risiko umum, seperti serangan denial-of-service (DoS). M AWS WAF enerapkan ke API ini dapat membantu melindungi aplikasi Anda dari serangan injeksi SQL dan skrip lintas situs (XSS). API Gateway juga mengimplementasikan throttling di AWS tingkat akun dan tingkat per-klien saat kunci API digunakan.

Dalam banyak kasus, fungsionalitas yang disediakan oleh API yang tidak diautentikasi dapat dicapai dengan pendekatan alternatif. Misalnya, aplikasi web mungkin menyediakan daftar toko ritel pelanggan dari tabel DynamoDB kepada pengguna yang tidak masuk. Permintaan ini mungkin berasal dari aplikasi web frontend atau dari sumber lain yang memanggil titik akhir URL. Diagram ini membandingkan tiga solusi:



1. API yang tidak diautentikasi ini dapat dipanggil oleh siapa saja di internet. Dalam serangan penolakan layanan, dimungkinkan untuk menghabiskan batas pelambatan API, konkurensi Lambda, atau kapasitas baca yang disediakan DynamoDB pada tabel yang mendasarinya.
2. Distri CloudFront busi di depan titik akhir API dengan konfigurasi time-to-live (TTL) yang sesuai akan menyerap sebagian besar lalu lintas dalam serangan DoS, tanpa mengubah solusi yang mendasari untuk mengambil data.
3. Atau, untuk data statis yang jarang berubah, CloudFront distribusi dapat melayani data' dari bucket Amazon S3.

Menggunakan penandatanganan kode untuk memverifikasi integritas kode dengan Lambda

Penandatanganan kode membantu memastikan bahwa hanya kode tepercaya yang diterapkan ke fungsi Lambda Anda. Dengan menggunakan AWS Signer, Anda dapat membuat paket kode yang ditandatangani secara digital untuk fungsi Anda. Saat Anda menambahkan konfigurasi penandatanganan kode ke fungsi, Lambda memverifikasi bahwa semua penerapan kode baru ditandatangani oleh sumber tepercaya. Karena pemeriksaan validasi penandatanganan kode berjalan pada waktu penerapan, tidak ada dampak pada eksekusi fungsi. Untuk informasi tentang mengelola kebijakan penandatanganan kode, lihatthe section called “Izin”.

> **Note**
>
> Penandatanganan kode berlaku untuk Lambda Event Functions yang digunakan sebagai paket gambar.zip atau container. Ini tidak digunakan oleh Lambda Web Functions atau Lambda MicroVMS.

> **Important**
>
> Konfigurasi penandatanganan kode hanya mencegah penerapan baru kode yang tidak ditandatangani. Jika Anda menambahkan konfigurasi penandatanganan kode ke fungsi yang sudah ada yang memiliki kode yang tidak ditandatangani, kode tersebut akan terus berjalan hingga Anda menerapkan paket kode baru.

Saat Anda mengaktifkan penandatanganan kode untuk suatu fungsi, lapisan apa pun yang Anda tambahkan ke fungsi juga harus ditandatangani oleh profil penandatanganan yang diizinkan.

Tidak ada biaya tambahan untuk penggunaan AWS Signer atau penandatanganan kode untuk AWS Lambda.

Validasi tanda tangan

Lambda melakukan pemeriksaan validasi berikut saat Anda men-deploy paket kode yang ditandatangani ke fungsi Anda:

1. Integritas: Memvalidasi bahwa paket kode belum dimodifikasi sejak ditandatangani. Lambda membandingkan hash paket dengan hash dari tanda tangan.
2. Kedaluwarsa: Memvalidasi bahwa tanda tangan paket kode belum kedaluwarsa.
3. Ketidakcocokan: Memvalidasi bahwa paket kode ditandatangani dengan profil penandatanganan yang diizinkan
4. Pencabutan: Memvalidasi bahwa tanda tangan paket kode belum dicabut.

Saat membuat konfigurasi penandatanganan kode, Anda dapat menggunakan UntrustedArtifactOnDeploymentparameter untuk menentukan respons Lambda jika pemeriksaan kedaluwarsa, ketidakcocokan, atau pencabutan gagal. Anda dapat memilih salah satu dari tindakan ini:

- `Warn`: Ini adalah pengaturan default. Lambda memungkinkan penyebaran paket kode, tetapi mengeluarkan peringatan. Lambda mengeluarkan CloudWatch metrik Amazon baru (`SignatureValidationErrors`) dan juga menyimpan peringatan di log. CloudTrail
- `Enforce`Lambda mengeluarkan peringatan (sama seperti untuk `Warn` tindakan) dan memblokir penerapan paket kode.

Topik

- Membuat konfigurasi penandatanganan kode untuk Lambda
- Mengonfigurasi kebijakan IAM untuk konfigurasi penandatanganan kode Lambda
- Menggunakan tag pada konfigurasi penandatanganan kode

Membuat konfigurasi penandatanganan kode untuk Lambda

Untuk mengaktifkan penandatanganan kode untuk fungsi, Anda membuat konfigurasi penandatanganan kode dan melampirkannya ke fungsi. Konfigurasi penandatanganan kode menetapkan daftar profil penandatanganan yang diizinkan dan tindakan kebijakan yang harus diambil jika pemeriksaan validasi gagal.

> **Note**
>
> Fungsi yang ditentukan sebagai gambar kontainer tidak mendukung penandatanganan kode.

Bagian-bagian

- Prasyarat konfigurasi
- Membuat konfigurasi penandatanganan kode
- Mengaktifkan penandatanganan kode untuk fungsi

Prasyarat konfigurasi

Sebelum Anda dapat mengonfigurasi penandatanganan kode untuk fungsi Lambda, gunakan AWS Penandatangan untuk melakukan hal berikut:

- Buat satu atau lebih profil penandatanganan.

- Gunakan profil penandatanganan untuk membuat paket kode yang ditandatangani untuk fungsi Anda.

Membuat konfigurasi penandatanganan kode

Konfigurasi penandatanganan kode menetapkan daftar profil penandatanganan yang diizinkan dan kebijakan validasi tanda tangan.

Untuk membuat konfigurasi penandatanganan kode (konsol)

1. Buka Halaman konfigurasi penandatanganan kode di konsol Lambda.
2. Pilih Buat konfigurasi.
3. Untuk Deskripsi, masukkan nama deskriptif untuk konfigurasi.
4. Di bawah Profil penandatanganan, tambahkan hingga 20 profil penandatanganan ke konfigurasi.
    a. Untuk ARN versi profil penandatanganan, pilih Amazon Resource Name (ARN) dari versi profil, atau masukkan ARN.
    b. Untuk menambahkan profil penandatanganan tambahan, pilih Tambahkan profil penandatanganan.
5. Di bawah Kebijakan validasi tanda tangan, pilih Warn atau Enforce.
6. Pilih Buat konfigurasi.

Mengaktifkan penandatanganan kode untuk fungsi

Untuk mengaktifkan penandatanganan kode untuk suatu fungsi, tambahkan konfigurasi penandatanganan kode ke fungsi.

> **Important**
>
> Konfigurasi penandatanganan kode hanya mencegah penerapan baru kode yang tidak ditandatangani. Jika Anda menambahkan konfigurasi penandatanganan kode ke fungsi yang sudah ada yang memiliki kode yang tidak ditandatangani, kode tersebut akan terus berjalan hingga Anda menerapkan paket kode baru.

Untuk mengaitkan konfigurasi penandatanganan kode dengan fungsi (konsol)

1. Buka halaman Fungsi Acara konsol Lambda.
2. Pilih fungsi yang ingin Anda aktifkan penandatanganan kodenya.
3. Buka tab Konfigurasi.
4. Gulir ke bawah dan pilih Penandatanganan kode.
5. Pilih Edit.
6. Di Edit penandatanganan kode, pilih konfigurasi penandatanganan kode untuk fungsi ini.
7. Pilih Simpan.

Mengonfigurasi kebijakan IAM untuk konfigurasi penandatanganan kode Lambda

Untuk memberikan izin bagi pengguna untuk mengakses operasi API penandatanganan kode Lambda, lampirkan satu atau beberapa pernyataan kebijakan ke kebijakan pengguna. Untuk informasi selengkapnya tentang kebijakan pengguna, lihat Identity-based Kebijakan IAM untuk Lambda.

Contoh pernyataan kebijakan berikut memberikan izin untuk membuat, memperbarui, dan mengambil konfigurasi penandatanganan kode.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
          "lambda:CreateCodeSigningConfig",
          "lambda:UpdateCodeSigningConfig",
          "lambda:GetCodeSigningConfig"
        ],
      "Resource": "*"
    }
  ]
}
```

Administrator dapat menggunakan kunci syarat `CodeSigningConfigArn` untuk menentukan konfigurasi penandatanganan kode yang harus digunakan developer untuk membuat atau memperbarui fungsi Anda.

Contoh pernyataan kebijakan berikut memberikan izin untuk membuat fungsi. Pernyataan kebijakan mencakup syarat `lambda:CodeSigningConfigArn` untuk menentukan konfigurasi penandatanganan kode yang diizinkan. Lambda memblokir permintaan `CreateFunction` API jika CodeSigningConfigArnparameter hilang atau tidak cocok dengan nilai dalam kondisi.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "AllowReferencingCodeSigningConfig",
      "Effect": "Allow",
      "Action": [
        "lambda:CreateFunction"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "lambda:CodeSigningConfigArn": "arn:aws:lambda:us-
east-1:111122223333:code-signing-config:csc-0d4518bd353a0a7c6"
        }
      }
    }
  ]
}
```

Menggunakan tag pada konfigurasi penandatanganan kode

Anda dapat menandai konfigurasi penandatanganan kode untuk mengatur dan mengelola sumber daya Anda. Tag adalah pasangan nilai kunci bentuk bebas yang terkait dengan sumber daya Anda yang didukung secara keseluruhan. Layanan AWS Untuk informasi selengkapnya tentang kasus penggunaan tag, lihat Strategi penandaan umum di AWS Sumber Daya Penandaan dan Panduan Editor Tag.

Anda dapat menggunakan AWS Lambda API untuk melihat dan memperbarui tag. Anda juga dapat melihat dan memperbarui tag sambil mengelola konfigurasi penandatanganan kode tertentu di konsol Lambda.

Bagian-bagian

- Izin diperlukan untuk bekerja dengan tag
- Menggunakan tag dengan konsol Lambda
- Menggunakan tag dengan AWS CLI

Izin diperlukan untuk bekerja dengan tag

Untuk mengizinkan identitas AWS Identity and Access Management (IAM) (pengguna, grup, atau peran) membaca atau menyetel tag pada sumber daya, berikan izin yang sesuai:

- lambda: ListTags —Ketika sumber daya memiliki tag, berikan izin ini kepada siapa saja yang perlu memanggilnya`ListTags`. Untuk Fungsi Acara yang ditandai, izin ini juga diperlukan untuk`GetFunction`.
- lambda: TagResource —Berikan izin ini kepada siapa saja yang perlu memanggil `TagResource` atau melakukan tag saat membuat.

Secara opsional, pertimbangkan untuk memberikan UntagResource izin lambda: juga untuk mengizinkan `UntagResource` panggilan ke sumber daya.

Untuk informasi selengkapnya, lihat Identity-based Kebijakan IAM untuk Lambda.

Menggunakan tag dengan konsol Lambda

Anda dapat menggunakan konsol Lambda untuk membuat konfigurasi penandatanganan kode yang memiliki tag, menambahkan tag ke konfigurasi penandatanganan kode yang ada, dan memfilter konfigurasi penandatanganan kode berdasarkan tag.

Untuk menambahkan tag saat Anda membuat konfigurasi penandatanganan kode

1. Buka konfigurasi penandatanganan Kode di konsol Lambda.
2. Dari header panel konten, Pilih Buat konfigurasi.

3. Di bagian di bagian atas panel konten, atur konfigurasi penandatanganan kode Anda. Untuk informasi selengkapnya tentang mengonfigurasi konfigurasi penandatanganan kode, lihat. the section called “Penandatanganan kode”
4. Di bagian Tag, pilih Tambahkan tag baru.
5. Di bidang Key, masukkan kunci tag Anda. Untuk informasi tentang pembatasan penandaan, lihat Batas dan persyaratan penamaan tag di Panduan AWS Sumber Daya Penandaan dan Editor Tag.
6. Pilih Buat konfigurasi.

Untuk menambahkan tag ke konfigurasi penandatanganan kode yang ada

1. Buka konfigurasi penandatanganan Kode di konsol Lambda.
2. Pilih nama konfigurasi penandatanganan kode Anda.
3. Pada tab di bawah panel Detail, pilih Tag.
4. Pilih Kelola tanda.
5. Pilih Tambahkan tag baru.
6. Di bidang Key, masukkan kunci tag Anda. Untuk informasi tentang pembatasan penandaan, lihat Batas dan persyaratan penamaan tag di Panduan AWS Sumber Daya Penandaan dan Editor Tag.
7. Pilih Simpan.

Untuk memfilter konfigurasi penandatanganan kode berdasarkan tag

1. Buka konfigurasi penandatanganan Kode di konsol Lambda.
2. Pilih kotak pencarian.
3. Dari daftar dropdown, pilih tag Anda dari bawah subjudul Tag.
4. Pilih Gunakan: “tag-name” untuk melihat semua konfigurasi penandatanganan kode yang ditandai dengan kunci ini, atau pilih Operator untuk memfilter lebih lanjut berdasarkan nilai.
5. Pilih nilai tag Anda untuk difilter dengan kombinasi kunci tag dan nilai.

Kotak pencarian juga mendukung pencarian kunci tag. Masukkan nama kunci untuk menemukannya dalam daftar.

Menggunakan tag dengan AWS CLI

Anda dapat menambahkan dan menghapus tag pada resource Lambda yang ada, termasuk konfigurasi penandatanganan kode, dengan API Lambda. Anda juga dapat menambahkan tag saat membuat konfigurasi penandatanganan kode agar sumber daya tetap diberi tag sepanjang siklus hidupnya.

Memperbarui tag dengan API tag Lambda

Anda dapat menambahkan dan menghapus tag untuk sumber daya Lambda yang didukung melalui operasi TagResourcedan UntagResourceAPI.

Anda dapat memanggil operasi ini menggunakan AWS CLI. Untuk menambahkan tag ke sumber daya yang ada, gunakan `tag-resource` perintah. Contoh ini menambahkan dua tag, satu dengan kunci *`Department`* dan satu dengan kunci*`CostCenter`*.

```
aws lambda tag-resource \
--resource arn:aws:lambda:us-east-2:123456789012:resource-type:my-resource \
--tags Department=Marketing,CostCenter=1234ABCD
```

Untuk menghapus tanda, gunakan perintah `untag-resource`. Contoh ini menghapus tag dengan kunci*`Department`*.

```
aws lambda untag-resource --resource arn:aws:lambda:us-east-1:123456789012:resource-
type:resource-identifier \
--tag-keys Department
```

Menambahkan tag saat membuat konfigurasi penandatanganan kode

Untuk membuat konfigurasi penandatanganan kode Lambda baru dengan tag, gunakan operasi CreateCodeSigningConfigAPI. Tentukan parameter `Tags`. Anda dapat memanggil operasi ini dengan `create-code-signing-config` AWS CLI perintah dan `--tags` opsi. Untuk informasi selengkapnya tentang perintah CLI, lihat create-code-signing-config di Command Reference.AWS CLI

Sebelum menggunakan `Tags` parameter dengan`CreateCodeSigningConfig`, pastikan bahwa peran Anda memiliki izin untuk menandai sumber daya bersama izin yang biasa diperlukan untuk operasi ini. Untuk informasi selengkapnya tentang izin untuk menandai, lihat. the section called “Izin diperlukan untuk bekerja dengan tag”

Melihat tag dengan API tag Lambda

Untuk melihat tag yang diterapkan ke sumber daya Lambda tertentu, gunakan operasi `ListTags` API. Untuk informasi selengkapnya, lihat ListTags.

Anda dapat memanggil operasi ini dengan `list-tags` AWS CLI perintah dengan memberikan ARN (Nama Sumber Daya Amazon).

```
aws lambda list-tags --resource arn:aws:lambda:us-east-1:123456789012:resource-type:resource-identifier
```

Memfilter sumber daya berdasarkan tag

Anda dapat menggunakan operasi AWS Resource Groups Tagging API GetResourcesAPI untuk memfilter sumber daya berdasarkan tag. `GetResources`Operasi menerima hingga 10 filter, dengan setiap filter berisi kunci tag dan hingga 10 nilai tag. Anda `GetResources` menyediakan `ResourceType` untuk memfilter berdasarkan jenis sumber daya tertentu.

Anda dapat memanggil operasi ini menggunakan `get-resources` AWS CLI perintah. Untuk contoh penggunaan`get-resources`, lihat mendapatkan sumber daya di Referensi Perintah AWS CLI.

Menggunakan Lambda dengan AWS SDK

AWS perangkat pengembangan perangkat lunak (SDK) tersedia untuk banyak bahasa pemrograman populer. Setiap SDK menyediakan API, contoh kode, dan dokumentasi yang memudahkan developer untuk membangun aplikasi dalam bahasa pilihan mereka.

Dokumentasi SDK	Contoh kode
AWS SDK untuk C++	AWS SDK untuk C++ contoh kode
AWS CLI	AWS CLI contoh kode
AWS SDK untuk Go	AWS SDK untuk Go contoh kode
AWS SDK untuk Java	AWS SDK untuk Java contoh kode
AWS SDK untuk JavaScript	AWS SDK untuk JavaScript contoh kode
AWS SDK untuk Kotlin	AWS SDK untuk Kotlin contoh kode
AWS SDK untuk .NET	AWS SDK untuk .NET contoh kode
AWS SDK untuk PHP	AWS SDK untuk PHP contoh kode
Alat AWS untuk PowerShell	Alat AWS untuk PowerShell contoh kode
AWS SDK untuk Python v4	AWS SDK untuk contoh kode Python v4
AWS SDK untuk Python (Boto3)	AWS SDK untuk Python (Boto3) contoh kode
AWS SDK untuk Ruby	AWS SDK untuk Ruby contoh kode
AWS SDK for Rust	AWS SDK for Rust contoh kode
AWS SDK untuk SAP ABAP	AWS SDK untuk SAP ABAP contoh kode
AWS SDK for Swift	AWS SDK for Swift contoh kode

Untuk contoh khusus untuk Lambda, lihat Contoh kode untuk menggunakan Lambda AWS SDK.

Ketersediaan contoh

Tidak dapat menemukan apa yang Anda butuhkan? Minta contoh kode menggunakan tautan Berikan umpan balik di bagian bawah halaman ini.

Contoh kode untuk menggunakan Lambda AWS SDK

Contoh kode berikut menunjukkan cara menggunakan Lambda dengan AWS kit pengembangan perangkat lunak (SDK).

Dasar-dasar adalah contoh kode yang menunjukkan kepada Anda bagaimana melakukan operasi penting dalam suatu layanan.

Tindakan merupakan kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Sementara tindakan menunjukkan cara memanggil fungsi layanan individual, Anda dapat melihat tindakan dalam konteks dalam skenario terkait.

Skenario adalah contoh kode yang menunjukkan kepada Anda cara menyelesaikan tugas tertentu dengan memanggil beberapa fungsi dalam layanan atau digabungkan dengan yang lain Layanan AWS.

AWS kontribusi masyarakat adalah contoh yang dibuat dan dikelola oleh beberapa tim di seluruh AWS. Untuk memberikan umpan balik, gunakan mekanisme yang disediakan di repositori tertaut.

Untuk daftar lengkap AWS Panduan pengembang SDK dan contoh kode, lihat Menggunakan Lambda dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Contoh kode menggunakan SDK

- Contoh dasar untuk menggunakan Lambda AWS SDK
  - Halo Lambda
  - Pelajari dasar-dasar Lambda dengan AWS SDK
  - Tindakan untuk Lambda menggunakan AWS SDK
    - Gunakan CreateAlias dengan CLI
    - Gunakan CreateFunction dengan AWS SDK atau CLI
    - Gunakan DeleteAlias dengan CLI
    - Gunakan DeleteFunction dengan AWS SDK atau CLI
    - Gunakan DeleteFunctionConcurrency dengan CLI
    - Gunakan DeleteProvisionedConcurrencyConfig dengan CLI
    - Gunakan GetAccountSettings dengan CLI
    - Gunakan GetAlias dengan CLI

- Gunakan GetFunction dengan AWS SDK atau CLI
- Gunakan GetFunctionConcurrency dengan CLI
- Gunakan GetFunctionConfiguration dengan CLI
- Gunakan GetPolicy dengan CLI
- Gunakan GetProvisionedConcurrencyConfig dengan CLI
- Gunakan Memanggil dengan AWS SDK atau CLI
- Gunakan ListFunctions dengan AWS SDK atau CLI
- Gunakan ListProvisionedConcurrencyConfigs dengan CLI
- Gunakan ListTags dengan CLI
- Gunakan ListVersionsByFunction dengan CLI
- Gunakan PublishVersion dengan CLI
- Gunakan PutFunctionConcurrency dengan CLI
- Gunakan PutProvisionedConcurrencyConfig dengan CLI
- Gunakan RemovePermission dengan CLI
- Gunakan TagResource dengan CLI
- Gunakan UntagResource dengan CLI
- Gunakan UpdateAlias dengan CLI
- Gunakan UpdateFunctionCode dengan AWS SDK atau CLI
- Gunakan UpdateFunctionConfiguration dengan AWS SDK atau CLI

- Skenario untuk Lambda menggunakan AWS SDK
  - Secara otomatis mengonfirmasi pengguna Amazon Cognito yang dikenal dengan fungsi Lambda menggunakan AWS SDK
  - Secara otomatis memigrasikan pengguna Amazon Cognito yang dikenal dengan fungsi Lambda menggunakan AWS SDK
  - Buat API Gateway REST API untuk melacak COVID-19 data
  - Membuat API REST pustaka peminjaman
  - Membuat aplikasi messenger dengan Step Functions
  - Buat API rest dengan integrasi proxy fungsi
  - Membuat aplikasi manajemen aset foto yang memungkinkan pengguna mengelola foto menggunakan label
  - Membuat aplikasi obrolan websocket dengan API Gateway

- Buat aplikasi yang menganalisis umpan balik pelanggan dan mensintesis audio
- Membuat dasbor pemantauan dengan nama fungsi sebagai variabel
- Membuat fungsi tanpa server pertama Anda
- Menginvokasi fungsi Lambda dari browser
- Ubah data untuk aplikasi Anda dengan S3 Object Lambda
- Menggunakan API Gateway untuk menginvokasi fungsi Lambda
- Menggunakan Step Functions untuk menginvokasi fungsi Lambda
- Menggunakan peristiwa terjadwal untuk menginvokasi fungsi Lambda
- Gunakan Amazon Neptune API untuk mengembangkan fungsi Lambda yang menanyakan data grafik
- Menggunakan variabel properti dalam dasbor pemantauan untuk memantau beberapa fungsi tanpa server
- Tulis data aktivitas khusus dengan fungsi Lambda setelah otentikasi pengguna Amazon Cognito menggunakan AWS SDK

- Contoh tanpa server untuk Lambda
  - Menghubungkan ke database Amazon RDS dalam fungsi Lambda
  - Memanggil fungsi Lambda dari pemicu Kinesis
  - Memanggil fungsi Lambda dari pemicu DynamoDB
  - Memanggil fungsi Lambda dari pemicu Amazon DocumentDB
  - Memanggil fungsi Lambda dari pemicu Amazon MSK
  - Menginvokasi fungsi Lambda dari pemicu Amazon S3
  - Memanggil fungsi Lambda dari pemicu Amazon SNS
  - Memanggil fungsi Lambda dari pemicu Amazon SQS
  - Melaporkan kegagalan item batch untuk fungsi Lambda dengan pemicu Kinesis
  - Melaporkan kegagalan item batch untuk fungsi Lambda dengan pemicu DynamoDB
  - Melaporkan kegagalan item batch untuk fungsi Lambda dengan pemicu Amazon SQS
- AWS kontribusi komunitas untuk Lambda
  - Membangun dan menguji aplikasi tanpa server

Contoh dasar untuk menggunakan Lambda AWS SDK

Contoh kode berikut menunjukkan cara menggunakan dasar-dasar AWS Lambda bersama AWS SDK.

Contoh

- Halo Lambda
- Pelajari dasar-dasar Lambda dengan AWS SDK
- Tindakan untuk Lambda menggunakan AWS SDK
  - Gunakan CreateAlias dengan CLI
  - Gunakan CreateFunction dengan AWS SDK atau CLI
  - Gunakan DeleteAlias dengan CLI
  - Gunakan DeleteFunction dengan AWS SDK atau CLI
  - Gunakan DeleteFunctionConcurrency dengan CLI
  - Gunakan DeleteProvisionedConcurrencyConfig dengan CLI
  - Gunakan GetAccountSettings dengan CLI
  - Gunakan GetAlias dengan CLI
  - Gunakan GetFunction dengan AWS SDK atau CLI
  - Gunakan GetFunctionConcurrency dengan CLI
  - Gunakan GetFunctionConfiguration dengan CLI
  - Gunakan GetPolicy dengan CLI
  - Gunakan GetProvisionedConcurrencyConfig dengan CLI
  - Gunakan Memanggil dengan AWS SDK atau CLI
  - Gunakan ListFunctions dengan AWS SDK atau CLI
  - Gunakan ListProvisionedConcurrencyConfigs dengan CLI
  - Gunakan ListTags dengan CLI
  - Gunakan ListVersionsByFunction dengan CLI
  - Gunakan PublishVersion dengan CLI
  - Gunakan PutFunctionConcurrency dengan CLI
  - Gunakan PutProvisionedConcurrencyConfig dengan CLI
  - Gunakan RemovePermission dengan CLI
  - Gunakan TagResource dengan CLI

- Gunakan UntagResource dengan CLI
- Gunakan UpdateAlias dengan CLI
- Gunakan UpdateFunctionCode dengan AWS SDK atau CLI
- Gunakan UpdateFunctionConfiguration dengan AWS SDK atau CLI

Halo Lambda

Contoh kode berikut menunjukkan cara memulai menggunakan Lambda.

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
namespace LambdaActions;

using Amazon.Lambda;

public class HelloLambda
{
    static async Task Main(string[] args)
    {
        var lambdaClient = new AmazonLambdaClient();

        Console.WriteLine("Hello AWS Lambda");
        Console.WriteLine("Let's get started with AWS Lambda by listing your
 existing Lambda functions:");

        var response = await lambdaClient.ListFunctionsAsync();
        response.Functions.ForEach(function =>
        {

 Console.WriteLine($"{function.FunctionName}\t{function.Description}");
        });
    }
```

```
}
```

- Untuk detail API, lihat ListFunctions di AWS SDK untuk .NET Referensi API.

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Kode untuk CMakeLists.txt File CMake.

```
# Set the minimum required version of CMake for this project.
cmake_minimum_required(VERSION 3.13)

# Set the AWS service components used by this project.
set(SERVICE_COMPONENTS lambda)

# Set this project's name.
project("hello_lambda")

# Set the C++ standard to use to build this target.
# At least C++ 11 is required for the AWS SDK for C++.
set(CMAKE_CXX_STANDARD 11)

# Use the MSVC variable to determine if this is a Windows build.
set(WINDOWS_BUILD ${MSVC})

if (WINDOWS_BUILD) # Set the location where CMake can find the installed
 libraries for the AWS SDK.
    string(REPLACE ";" "/aws-cpp-sdk-all;" SYSTEM_MODULE_PATH
 "${CMAKE_SYSTEM_PREFIX_PATH}/aws-cpp-sdk-all")
    list(APPEND CMAKE_PREFIX_PATH ${SYSTEM_MODULE_PATH})
endif ()

# Find the AWS SDK for C++ package.
```

```
find_package(AWSSDK REQUIRED COMPONENTS ${SERVICE_COMPONENTS})

if (WINDOWS_BUILD AND AWSSDK_INSTALL_AS_SHARED_LIBS)
     # Copy relevant AWS SDK for C++ libraries into the current binary directory
 for running and debugging.

     # set(BIN_SUB_DIR "/Debug") # if you are building from the command line you
 may need to uncomment this
                                     # and set the proper subdirectory to the
 executables' location.

     AWSSDK_CPY_DYN_LIBS(SERVICE_COMPONENTS ""
 ${CMAKE_CURRENT_BINARY_DIR}${BIN_SUB_DIR})
endif ()

add_executable(${PROJECT_NAME}
        hello_lambda.cpp)

target_link_libraries(${PROJECT_NAME}
        ${AWSSDK_LINK_LIBRARIES})
```

Kode untuk file sumber hello_lambda.cpp.

```
#include <aws/core/Aws.h>
#include <aws/lambda/LambdaClient.h>
#include <aws/lambda/model/ListFunctionsRequest.h>
#include <iostream>

/*
 *  A "Hello Lambda" starter application which initializes an AWS Lambda (Lambda)
 client and lists the Lambda functions.
 *
 *  main function
 *
 *  Usage: 'hello_lambda'
 *
 */

int main(int argc, char **argv) {
    Aws::SDKOptions options;
    // Optionally change the log level for debugging.
//   options.loggingOptions.logLevel = Utils::Logging::LogLevel::Debug;
```

```
    Aws::InitAPI(options); // Should only be called once.
    int result = 0;
    {
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

        Aws::Lambda::LambdaClient lambdaClient(clientConfig);
        std::vector<Aws::String> functions;
        Aws::String marker; // Used for pagination.

        do {
            Aws::Lambda::Model::ListFunctionsRequest request;
            if (!marker.empty()) {
                request.SetMarker(marker);
            }

            Aws::Lambda::Model::ListFunctionsOutcome outcome =
lambdaClient.ListFunctions(
                    request);

            if (outcome.IsSuccess()) {
                const Aws::Lambda::Model::ListFunctionsResult
&listFunctionsResult = outcome.GetResult();
                std::cout << listFunctionsResult.GetFunctions().size()
                          << " lambda functions were retrieved." << std::endl;

                for (const Aws::Lambda::Model::FunctionConfiguration
&functionConfiguration: listFunctionsResult.GetFunctions()) {
                    functions.push_back(functionConfiguration.GetFunctionName());
                    std::cout << functions.size() << "  "
                              << functionConfiguration.GetDescription() <<
std::endl;
                    std::cout << "   "
                              <<
Aws::Lambda::Model::RuntimeMapper::GetNameForRuntime(
                                      functionConfiguration.GetRuntime()) << ": "
                              << functionConfiguration.GetHandler()
                              << std::endl;
                }
                marker = listFunctionsResult.GetNextMarker();
            } else {
                std::cerr << "Error with Lambda::ListFunctions. "
                          << outcome.GetError().GetMessage()
```

```
                            << std::endl;
                result = 1;
                break;
            }
        } while (!marker.empty());
    }


    Aws::ShutdownAPI(options); // Should only be called once.
    return result;
}
```

- Untuk detail API, lihat ListFunctions di AWS SDK untuk C++ Referensi API.

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
package main

import (
 "context"
 "fmt"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/lambda"
)

// main uses the AWS SDK for Go (v2) to create an AWS Lambda client and list up
 to 10
// functions in your account.
// This example uses the default settings specified in your shared credentials
// and config files.
```

```
func main() {
 ctx := context.Background()
 sdkConfig, err := config.LoadDefaultConfig(ctx)
 if err != nil {
  fmt.Println("Couldn't load default configuration. Have you set up your AWS
 account?")
  fmt.Println(err)
  return
 }
 lambdaClient := lambda.NewFromConfig(sdkConfig)

 maxItems := 10
 fmt.Printf("Let's list up to %v functions for your account.\n", maxItems)
 result, err := lambdaClient.ListFunctions(ctx, &lambda.ListFunctionsInput{
  MaxItems: aws.Int32(int32(maxItems)),
 })
 if err != nil {
  fmt.Printf("Couldn't list functions for your account. Here's why: %v\n", err)
  return
 }
 if len(result.Functions) == 0 {
  fmt.Println("You don't have any functions!")
 } else {
  for _, function := range result.Functions {
   fmt.Printf("\t%v\n", *function.FunctionName)
  }
 }
}
```

- Untuk detail API, lihat ListFunctions di AWS SDK untuk Go Referensi API.

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /**
     * Lists the AWS Lambda functions associated with the current AWS account.
     *
     * @param awsLambda an instance of the {@link LambdaClient} class, which is
used to interact with the AWS Lambda service
     *
     * @throws LambdaException if an error occurs while interacting with the AWS
Lambda service
     */
    public static void listFunctions(LambdaClient awsLambda) {
        try {
            ListFunctionsResponse functionResult = awsLambda.listFunctions();
            List<FunctionConfiguration> list = functionResult.functions();
            for (FunctionConfiguration config : list) {
                System.out.println("The function name is " +
config.functionName());
            }

        } catch (LambdaException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
```

- Untuk detail API, lihat ListFunctions di AWS SDK for Java 2.x Referensi API.

JavaScript

SDK untuk JavaScript (V3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
import { LambdaClient, paginateListFunctions } from "@aws-sdk/client-lambda";

const client = new LambdaClient({});
```

```
export const helloLambda = async () => {
  const paginator = paginateListFunctions({ client }, {});
  const functions = [];

  for await (const page of paginator) {
    const funcNames = page.Functions.map((f) => f.FunctionName);
    functions.push(...funcNames);
  }

  console.log("Functions:");
  console.log(functions.join("\n"));
  return functions;
};
```

- Untuk detail API, lihat ListFunctions di AWS SDK untuk JavaScript Referensi API.

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
import boto3


def main():
    """
    List the Lambda functions in your AWS account.
    """
    # Create the Lambda client
    lambda_client = boto3.client("lambda")

    # Use the paginator to list the functions
    paginator = lambda_client.get_paginator("list_functions")
    response_iterator = paginator.paginate()
```

```
    print("Here are the Lambda functions in your account:")
    for page in response_iterator:
        for function in page["Functions"]:
            print(f"  {function['FunctionName']}")


if __name__ == "__main__":
    main()
```

- Untuk detail API, lihat ListFunctions di AWS Referensi API SDK untuk Python (Boto3).

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
require 'aws-sdk-lambda'

# Creates an AWS Lambda client using the default credentials and configuration
def lambda_client
  Aws::Lambda::Client.new
end

# Lists the Lambda functions in your AWS account, paginating the results if
 necessary
def list_lambda_functions
  lambda = lambda_client

  # Use a pagination iterator to list all functions
  functions = []
  lambda.list_functions.each_page do |page|
    functions.concat(page.functions)
  end
```

```
  # Print the name and ARN of each function
  functions.each do |function|
    puts "Function name: #{function.function_name}"
    puts "Function ARN: #{function.function_arn}"
    puts
  end

  puts "Total functions: #{functions.count}"
end

list_lambda_functions if __FILE__ == $PROGRAM_NAME
```

- Untuk detail API, lihat ListFunctions di AWS SDK untuk Ruby Referensi API.

Untuk daftar lengkap AWS Panduan pengembang SDK dan contoh kode, lihat Menggunakan Lambda dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Pelajari dasar-dasar Lambda dengan AWS SDK

Contoh-contoh kode berikut menunjukkan cara:

- Buat peran IAM dan fungsi Lambda, lalu unggah kode handler.
- Memanggil fungsi dengan satu parameter dan dapatkan hasil.
- Perbarui kode fungsi dan konfigurasikan dengan variabel lingkungan.
- Memanggil fungsi dengan parameter baru dan dapatkan hasil. Menampilkan log eksekusi yang dikembalikan.
- Buat daftar fungsi untuk akun Anda, lalu bersihkan sumber daya.

Untuk informasi lebih lanjut, lihat Buat fungsi Lambda dengan konsol.

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Buat metode yang melakukan tindakan Lambda.

```
namespace LambdaActions;

using Amazon.Lambda;
using Amazon.Lambda.Model;

/// <summary>
/// A class that implements AWS Lambda methods.
/// </summary>
public class LambdaWrapper
{
    private readonly IAmazonLambda _lambdaService;

    /// <summary>
    /// Constructor for the LambdaWrapper class.
    /// </summary>
    /// <param name="lambdaService">An initialized Lambda service client.</param>
    public LambdaWrapper(IAmazonLambda lambdaService)
    {
        _lambdaService = lambdaService;
    }

    /// <summary>
    /// Creates a new Lambda function.
    /// </summary>
    /// <param name="functionName">The name of the function.</param>
    /// <param name="s3Bucket">The Amazon Simple Storage Service (Amazon S3)
    /// bucket where the zip file containing the code is located.</param>
    /// <param name="s3Key">The Amazon S3 key of the zip file.</param>
    /// <param name="role">The Amazon Resource Name (ARN) of a role with the
    /// appropriate Lambda permissions.</param>
    /// <param name="handler">The name of the handler function.</param>
```

```
    /// <returns>The Amazon Resource Name (ARN) of the newly created
    /// Lambda function.</returns>
    public async Task<string> CreateLambdaFunctionAsync(
        string functionName,
        string s3Bucket,
        string s3Key,
        string role,
        string handler)
    {
        // Defines the location for the function code.
        // S3Bucket - The S3 bucket where the file containing
        //            the source code is stored.
        // S3Key    - The name of the file containing the code.
        var functionCode = new FunctionCode
        {
            S3Bucket = s3Bucket,
            S3Key = s3Key,
        };

        var createFunctionRequest = new CreateFunctionRequest
        {
            FunctionName = functionName,
            Description = "Created by the Lambda .NET API",
            Code = functionCode,
            Handler = handler,
            Runtime = Runtime.Dotnet6,
            Role = role,
        };

        var reponse = await
 _lambdaService.CreateFunctionAsync(createFunctionRequest);
        return reponse.FunctionArn;
    }


    /// <summary>
    /// Delete an AWS Lambda function.
    /// </summary>
    /// <param name="functionName">The name of the Lambda function to
    /// delete.</param>
    /// <returns>A Boolean value that indicates the success of the action.</
returns>
    public async Task<bool> DeleteFunctionAsync(string functionName)
    {
```

```
        var request = new DeleteFunctionRequest
        {
            FunctionName = functionName,
        };

        var response = await _lambdaService.DeleteFunctionAsync(request);

        // A return value of NoContent means that the request was processed.
        // In this case, the function was deleted, and the return value
        // is intentionally blank.
        return response.HttpStatusCode == System.Net.HttpStatusCode.NoContent;
    }


    /// <summary>
    /// Gets information about a Lambda function.
    /// </summary>
    /// <param name="functionName">The name of the Lambda function for
    /// which to retrieve information.</param>
    /// <returns>Async Task.</returns>
    public async Task<FunctionConfiguration> GetFunctionAsync(string
functionName)
    {
        var functionRequest = new GetFunctionRequest
        {
            FunctionName = functionName,
        };

        var response = await _lambdaService.GetFunctionAsync(functionRequest);
        return response.Configuration;
    }


    /// <summary>
    /// Invoke a Lambda function.
    /// </summary>
    /// <param name="functionName">The name of the Lambda function to
    /// invoke.</param
    /// <param name="parameters">The parameter values that will be passed to the
function.</param>
    /// <returns>A System Threading Task.</returns>
    public async Task<string> InvokeFunctionAsync(
        string functionName,
        string parameters)
```

```
    {
        var payload = parameters;
        var request = new InvokeRequest
        {
            FunctionName = functionName,
            Payload = payload,
        };

        var response = await _lambdaService.InvokeAsync(request);
        MemoryStream stream = response.Payload;
        string returnValue =
 System.Text.Encoding.UTF8.GetString(stream.ToArray());
        return returnValue;
    }


    /// <summary>
    /// Get a list of Lambda functions.
    /// </summary>
    /// <returns>A list of FunctionConfiguration objects.</returns>
    public async Task<List<FunctionConfiguration>> ListFunctionsAsync()
    {
        var functionList = new List<FunctionConfiguration>();

        var functionPaginator =
            _lambdaService.Paginators.ListFunctions(new ListFunctionsRequest());
        await foreach (var function in functionPaginator.Functions)
        {
            functionList.Add(function);
        }

        return functionList;
    }


    /// <summary>
    /// Update an existing Lambda function.
    /// </summary>
    /// <param name="functionName">The name of the Lambda function to update.</
param>
    /// <param name="bucketName">The bucket where the zip file containing
    /// the Lambda function code is stored.</param>
    /// <param name="key">The key name of the source code file.</param>
    /// <returns>Async Task.</returns>
```

```
    public async Task UpdateFunctionCodeAsync(
        string functionName,
        string bucketName,
        string key)
    {
        var functionCodeRequest = new UpdateFunctionCodeRequest
        {
            FunctionName = functionName,
            Publish = true,
            S3Bucket = bucketName,
            S3Key = key,
        };

        var response = await
_lambdaService.UpdateFunctionCodeAsync(functionCodeRequest);
        Console.WriteLine($"The Function was last modified at
{response.LastModified}.");
    }


    /// <summary>
    /// Update the code of a Lambda function.
    /// </summary>
    /// <param name="functionName">The name of the function to update.</param>
    /// <param name="functionHandler">The code that performs the function's
actions.</param>
    /// <param name="environmentVariables">A dictionary of environment
variables.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> UpdateFunctionConfigurationAsync(
        string functionName,
        string functionHandler,
        Dictionary<string, string> environmentVariables)
    {
        var request = new UpdateFunctionConfigurationRequest
        {
            Handler = functionHandler,
            FunctionName = functionName,
            Environment = new Amazon.Lambda.Model.Environment { Variables =
environmentVariables },
        };

        var response = await
_lambdaService.UpdateFunctionConfigurationAsync(request);
```

```
        Console.WriteLine(response.LastModified);

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


}
```

Buat fungsi yang menjalankan skenario.

```
global using System.Threading.Tasks;
global using Amazon.IdentityManagement;
global using Amazon.Lambda;
global using LambdaActions;
global using LambdaScenarioCommon;
global using Microsoft.Extensions.DependencyInjection;
global using Microsoft.Extensions.Hosting;
global using Microsoft.Extensions.Logging;
global using Microsoft.Extensions.Logging.Console;
global using Microsoft.Extensions.Logging.Debug;


using Amazon.Lambda.Model;
using Microsoft.Extensions.Configuration;

namespace LambdaBasics;

public class LambdaBasics
{
    private static ILogger logger = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for the Amazon service.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
 LogLevel.Information)
```

```
                    .AddFilter<ConsoleLoggerProvider>("Microsoft",
LogLevel.Trace))
            .ConfigureServices((_, services) =>
            services.AddAWSService<IAmazonLambda>()
            .AddAWSService<IAmazonIdentityManagementService>()
            .AddTransient<LambdaWrapper>()
            .AddTransient<LambdaRoleWrapper>()
            .AddTransient<UIWrapper>()
        )
        .Build();

        var configuration = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddJsonFile("settings.json") // Load test settings from .json file.
            .AddJsonFile("settings.local.json",
            true) // Optionally load local settings.
        .Build();


        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<LambdaBasics>();

        var lambdaWrapper = host.Services.GetRequiredService<LambdaWrapper>();
        var lambdaRoleWrapper =
host.Services.GetRequiredService<LambdaRoleWrapper>();
        var uiWrapper = host.Services.GetRequiredService<UIWrapper>();

        string functionName = configuration["FunctionName"]!;
        string roleName = configuration["RoleName"]!;
        string policyDocument = "{" +
            " \"Version\": \"2012-10-17\"," +
            " \"Statement\": [ " +
            "    {" +
            "        \"Effect\": \"Allow\"," +
            "        \"Principal\": {" +
            "            \"Service\": \"lambda.amazonaws.com\" " +
            "    }," +
            "        \"Action\": \"sts:AssumeRole\" " +
            "    }" +
            "]" +
        "}";

        var incrementHandler = configuration["IncrementHandler"];
        var calculatorHandler = configuration["CalculatorHandler"];
```

```
        var bucketName = configuration["BucketName"];
        var incrementKey = configuration["IncrementKey"];
        var calculatorKey = configuration["CalculatorKey"];
        var policyArn = configuration["PolicyArn"];

        uiWrapper.DisplayLambdaBasicsOverview();

        // Create the policy to use with the AWS Lambda functions and then attach
the
        // policy to a new role.
        var roleArn = await lambdaRoleWrapper.CreateLambdaRoleAsync(roleName,
policyDocument);

        Console.WriteLine("Waiting for role to become active.");
        uiWrapper.WaitABit(15, "Wait until the role is active before trying to
use it.");

        // Attach the appropriate AWS Identity and Access Management (IAM) role
policy to the new role.
        var success = await
lambdaRoleWrapper.AttachLambdaRolePolicyAsync(policyArn, roleName);
        uiWrapper.WaitABit(10, "Allow time for the IAM policy to be attached to
the role.");

        // Create the Lambda function using a zip file stored in an Amazon Simple
Storage Service
        // (Amazon S3) bucket.
        uiWrapper.DisplayTitle("Create Lambda Function");
        Console.WriteLine($"Creating the AWS Lambda function: {functionName}.");
        var lambdaArn = await lambdaWrapper.CreateLambdaFunctionAsync(
            functionName,
            bucketName,
            incrementKey,
            roleArn,
            incrementHandler);

        Console.WriteLine("Waiting for the new function to be available.");
        Console.WriteLine($"The AWS Lambda ARN is {lambdaArn}");

        // Get the Lambda function.
        Console.WriteLine($"Getting the {functionName} AWS Lambda function.");
        FunctionConfiguration config;
        do
        {
```

```
            config = await lambdaWrapper.GetFunctionAsync(functionName);
            Console.Write(".");
        }
        while (config.State != State.Active);

        Console.WriteLine($"\nThe function, {functionName} has been created.");
        Console.WriteLine($"The runtime of this Lambda function is
{config.Runtime}.");

        uiWrapper.PressEnter();

        // List the Lambda functions.
        uiWrapper.DisplayTitle("Listing all Lambda functions.");
        var functions = await lambdaWrapper.ListFunctionsAsync();
        DisplayFunctionList(functions);

        uiWrapper.DisplayTitle("Invoke increment function");
        Console.WriteLine("Now that it has been created, invoke the Lambda
increment function.");
        string? value;
        do
        {
            Console.Write("Enter a value to increment: ");
            value = Console.ReadLine();
        }
        while (string.IsNullOrEmpty(value));

        string functionParameters = "{" +
            "\"action\": \"increment\", " +
            "\"x\": \"" + value + "\"" +
        "}";
        var answer = await lambdaWrapper.InvokeFunctionAsync(functionName,
functionParameters);
        Console.WriteLine($"{value} + 1 = {answer}.");

        uiWrapper.DisplayTitle("Update function");
        Console.WriteLine("Now update the Lambda function code.");
        await lambdaWrapper.UpdateFunctionCodeAsync(functionName, bucketName,
calculatorKey);

        do
        {
            config = await lambdaWrapper.GetFunctionAsync(functionName);
            Console.Write(".");
```

```
        }
        while (config.LastUpdateStatus == LastUpdateStatus.InProgress);

        await lambdaWrapper.UpdateFunctionConfigurationAsync(
            functionName,
            calculatorHandler,
            new Dictionary<string, string> { { "LOG_LEVEL", "DEBUG" } });

        do
        {
            config = await lambdaWrapper.GetFunctionAsync(functionName);
            Console.Write(".");
        }
        while (config.LastUpdateStatus == LastUpdateStatus.InProgress);

        uiWrapper.DisplayTitle("Call updated function");
        Console.WriteLine("Now call the updated function...");

        bool done = false;

        do
        {
            string? opSelected;

            Console.WriteLine("Select the operation to perform:");
            Console.WriteLine("\t1. add");
            Console.WriteLine("\t2. subtract");
            Console.WriteLine("\t3. multiply");
            Console.WriteLine("\t4. divide");
            Console.WriteLine("\tOr enter \"q\" to quit.");
            Console.WriteLine("Enter the number (1, 2, 3, 4, or q) of the
operation you want to perform: ");
            do
            {
                Console.Write("Your choice? ");
                opSelected = Console.ReadLine();
            }
            while (opSelected == string.Empty);

            var operation = (opSelected) switch
            {
                "1" => "add",
                "2" => "subtract",
                "3" => "multiply",
```

```
                "4" => "divide",
                "q" => "quit",
                _ => "add",
            };

            if (operation == "quit")
            {
                done = true;
            }
            else
            {
                // Get two numbers and an action from the user.
                value = string.Empty;
                do
                {
                    Console.Write("Enter the first value: ");
                    value = Console.ReadLine();
                }
                while (value == string.Empty);

                string? value2;
                do
                {
                    Console.Write("Enter a second value: ");
                    value2 = Console.ReadLine();
                }
                while (value2 == string.Empty);

                functionParameters = "{" +
                    "\"action\": \"" + operation + "\", " +
                    "\"x\": \"" + value + "\"," +
                    "\"y\": \"" + value2 + "\"" +
                "}";

                answer = await lambdaWrapper.InvokeFunctionAsync(functionName,
functionParameters);
                Console.WriteLine($"The answer when we {operation} the two
numbers is: {answer}.");
            }

            uiWrapper.PressEnter();
        } while (!done);

        // Delete the function created earlier.
```

```
        uiWrapper.DisplayTitle("Clean up resources");
        // Detach the IAM policy from the IAM role.
        Console.WriteLine("First detach the IAM policy from the role.");
        success = await lambdaRoleWrapper.DetachLambdaRolePolicyAsync(policyArn,
roleName);
        uiWrapper.WaitABit(15, "Let's wait for the policy to be fully detached
from the role.");

        Console.WriteLine("Delete the AWS Lambda function.");
        success = await lambdaWrapper.DeleteFunctionAsync(functionName);
        if (success)
        {
            Console.WriteLine($"The {functionName} function was deleted.");
        }
        else
        {
            Console.WriteLine($"Could not remove the function {functionName}");
        }

        // Now delete the IAM role created for use with the functions
        // created by the application.
        Console.WriteLine("Now we can delete the role that we created.");
        success = await lambdaRoleWrapper.DeleteLambdaRoleAsync(roleName);
        if (success)
        {
            Console.WriteLine("The role has been successfully removed.");
        }
        else
        {
            Console.WriteLine("Couldn't delete the role.");
        }

        Console.WriteLine("The Lambda Scenario is now complete.");
        uiWrapper.PressEnter();

        // Displays a formatted list of existing functions returned by the
        // LambdaMethods.ListFunctions.
        void DisplayFunctionList(List<FunctionConfiguration> functions)
        {
            functions.ForEach(functionConfig =>
            {

Console.WriteLine($"{functionConfig.FunctionName}\t{functionConfig.Description}");
```

```
            });
        }
    }
}


namespace LambdaActions;

using Amazon.IdentityManagement;
using Amazon.IdentityManagement.Model;

public class LambdaRoleWrapper
{
    private readonly IAmazonIdentityManagementService _lambdaRoleService;

    public LambdaRoleWrapper(IAmazonIdentityManagementService lambdaRoleService)
    {
        _lambdaRoleService = lambdaRoleService;
    }

    /// <summary>
    /// Attach an AWS Identity and Access Management (IAM) role policy to the
    /// IAM role to be assumed by the AWS Lambda functions created for the
 scenario.
    /// </summary>
    /// <param name="policyArn">The Amazon Resource Name (ARN) of the IAM
 policy.</param>
    /// <param name="roleName">The name of the IAM role to attach the IAM policy
 to.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> AttachLambdaRolePolicyAsync(string policyArn, string
 roleName)
    {
        var response = await _lambdaRoleService.AttachRolePolicyAsync(new
 AttachRolePolicyRequest { PolicyArn = policyArn, RoleName = roleName });
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    /// <summary>
    /// Create a new IAM role.
    /// </summary>
    /// <param name="roleName">The name of the IAM role to create.</param>
    /// <param name="policyDocument">The policy document for the new IAM role.</
param>
```

```
    /// <returns>A string representing the ARN for newly created role.</returns>
    public async Task<string> CreateLambdaRoleAsync(string roleName, string
 policyDocument)
    {
        var request = new CreateRoleRequest
        {
            AssumeRolePolicyDocument = policyDocument,
            RoleName = roleName,
        };

        var response = await _lambdaRoleService.CreateRoleAsync(request);
        return response.Role.Arn;
    }

    /// <summary>
    /// Deletes an IAM role.
    /// </summary>
    /// <param name="roleName">The name of the role to delete.</param>
    /// <returns>A Boolean value indicating the success of the operation.</
returns>
    public async Task<bool> DeleteLambdaRoleAsync(string roleName)
    {
        var request = new DeleteRoleRequest
        {
            RoleName = roleName,
        };

        var response = await _lambdaRoleService.DeleteRoleAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    public async Task<bool> DetachLambdaRolePolicyAsync(string policyArn, string
 roleName)
    {
        var response = await _lambdaRoleService.DetachRolePolicyAsync(new
 DetachRolePolicyRequest { PolicyArn = policyArn, RoleName = roleName });
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
}


namespace LambdaScenarioCommon;

public class UIWrapper
```

```
{
    public readonly string SepBar = new('-', Console.WindowWidth);

    /// <summary>
    /// Show information about the AWS Lambda Basics scenario.
    /// </summary>
    public void DisplayLambdaBasicsOverview()
    {
        Console.Clear();

        DisplayTitle("Welcome to AWS Lambda Basics");
        Console.WriteLine("This example application does the following:");
        Console.WriteLine("\t1. Creates an AWS Identity and Access Management
 (IAM) role that will be assumed by the functions we create.");
        Console.WriteLine("\t2. Attaches an IAM role policy that has Lambda
 permissions.");
        Console.WriteLine("\t3. Creates a Lambda function that increments the
 value passed to it.");
        Console.WriteLine("\t4. Calls the increment function and passes a
 value.");
        Console.WriteLine("\t5. Updates the code so that the function is a simple
 calculator.");
        Console.WriteLine("\t6. Calls the calculator function with the values
 entered.");
        Console.WriteLine("\t7. Deletes the Lambda function.");
        Console.WriteLine("\t7. Detaches the IAM role policy.");
        Console.WriteLine("\t8. Deletes the IAM role.");
        PressEnter();
    }

    /// <summary>
    /// Display a message and wait until the user presses enter.
    /// </summary>
    public void PressEnter()
    {
        Console.Write("\nPress <Enter> to continue. ");
        _ = Console.ReadLine();
        Console.WriteLine();
    }

    /// <summary>
    /// Pad a string with spaces to center it on the console display.
    /// </summary>
    /// <param name="strToCenter">The string to be centered.</param>
```

```
    /// <returns>The padded string.</returns>
    public string CenterString(string strToCenter)
    {
        var padAmount = (Console.WindowWidth - strToCenter.Length) / 2;
        var leftPad = new string(' ', padAmount);
        return $"{leftPad}{strToCenter}";
    }

    /// <summary>
    /// Display a line of hyphens, the centered text of the title and another
    /// line of hyphens.
    /// </summary>
    /// <param name="strTitle">The string to be displayed.</param>
    public void DisplayTitle(string strTitle)
    {
        Console.WriteLine(SepBar);
        Console.WriteLine(CenterString(strTitle));
        Console.WriteLine(SepBar);
    }

    /// <summary>
    /// Display a countdown and wait for a number of seconds.
    /// </summary>
    /// <param name="numSeconds">The number of seconds to wait.</param>
    public void WaitABit(int numSeconds, string msg)
    {
        Console.WriteLine(msg);

        // Wait for the requested number of seconds.
        for (int i = numSeconds; i > 0; i--)
        {
            System.Threading.Thread.Sleep(1000);
            Console.Write($"{i}...");
        }

        PressEnter();
    }
}
```

Tentukan handler Lambda yang menambah angka.

```
using Amazon.Lambda.Core;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri

namespace LambdaIncrement;

public class Function
{

    /// <summary>
    /// A simple function increments the integer parameter.
    /// </summary>
    /// <param name="input">A JSON string containing an action, which must be
    /// "increment" and a string representing the value to increment.</param>
    /// <param name="context">The context object passed by Lambda containing
    /// information about invocation, function, and execution environment.</
param>
    /// <returns>A string representing the incremented value of the parameter.</
returns>
    public int FunctionHandler(Dictionary<string, string> input, ILambdaContext
 context)
    {
        if (input["action"] == "increment")
        {
            int inputValue = Convert.ToInt32(input["x"]);
            return inputValue + 1;
        }
        else
        {
            return 0;
        }
    }
}
```

Tentukan handler Lambda kedua yang melakukan operasi aritmatika.

```
using Amazon.Lambda.Core;
```

```
// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri

namespace LambdaCalculator;

public class Function
{

    /// <summary>
    /// A simple function that takes two number in string format and performs
    /// the requested arithmetic function.
    /// </summary>
    /// <param name="input">JSON data containing an action, and x and y values.
    /// Valid actions include: add, subtract, multiply, and divide.</param>
    /// <param name="context">The context object passed by Lambda containing
    /// information about invocation, function, and execution environment.</
param>
    /// <returns>A string representing the results of the calculation.</returns>
    public int FunctionHandler(Dictionary<string, string> input, ILambdaContext
 context)
    {
        var action = input["action"];
        int x = Convert.ToInt32(input["x"]);
        int y = Convert.ToInt32(input["y"]);
        int result;
        switch (action)
        {
            case "add":
                result = x + y;
                break;
            case "subtract":
                result = x - y;
                break;
            case "multiply":
                result = x * y;
                break;
            case "divide":
                if (y == 0)
                {
                    Console.Error.WriteLine("Divide by zero error.");
                    result = 0;
                }
```

```
                else
                    result = x / y;
                break;
            default:
                Console.Error.WriteLine($"{action} is not a valid operation.");
                result = 0;
                break;
        }
        return result;
    }
}
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK untuk .NET .
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Memanggil
  - ListFunctions
  - UpdateFunctionCode
  - UpdateFunctionConfiguration

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkap dan pelajari cara menyiapkan dan menjalankan di Repositori Contoh Kode AWS.

```
//! Get started with functions scenario.
/*!
 \param clientConfig: AWS client configuration.
 \return bool: Successful completion.
 */
```

```
bool AwsDoc::Lambda::getStartedWithFunctionsScenario(
        const Aws::Client::ClientConfiguration &clientConfig) {

    Aws::Lambda::LambdaClient client(clientConfig);

    // 1. Create an AWS Identity and Access Management (IAM) role for Lambda
 function.
    Aws::String roleArn;
    if (!getIamRoleArn(roleArn, clientConfig)) {
        return false;
    }

    // 2. Create a Lambda function.
    int seconds = 0;
    do {
        Aws::Lambda::Model::CreateFunctionRequest request;
        request.SetFunctionName(LAMBDA_NAME);
        request.SetDescription(LAMBDA_DESCRIPTION); // Optional.
#if USE_CPP_LAMBDA_FUNCTION
        request.SetRuntime(Aws::Lambda::Model::Runtime::provided_al2);
        request.SetTimeout(15);
        request.SetMemorySize(128);

        // Assume the AWS Lambda function was built in Docker with same
 architecture
        // as this code.
#if  defined(__x86_64__)
        request.SetArchitectures({Aws::Lambda::Model::Architecture::x86_64});
#elif defined(__aarch64__)
        request.SetArchitectures({Aws::Lambda::Model::Architecture::arm64});
#else
#error "Unimplemented architecture"
#endif // defined(architecture)
#else
        request.SetRuntime(Aws::Lambda::Model::Runtime::python3_9);
#endif
        request.SetRole(roleArn);
        request.SetHandler(LAMBDA_HANDLER_NAME);
        request.SetPublish(true);
        Aws::Lambda::Model::FunctionCode code;
        std::ifstream ifstream(INCREMENT_LAMBDA_CODE.c_str(),
                               std::ios_base::in | std::ios_base::binary);
        if (!ifstream.is_open()) {
```

```
            std::cerr << "Error opening file " << INCREMENT_LAMBDA_CODE << "." <<
 std::endl;

#if USE_CPP_LAMBDA_FUNCTION
            std::cerr
                    << "The cpp Lambda function must be built following the
 instructions in the cpp_lambda/README.md file. "
                    << std::endl;
#endif
            deleteIamRole(clientConfig);
            return false;
        }

        Aws::StringStream buffer;
        buffer << ifstream.rdbuf();

        code.SetZipFile(Aws::Utils::ByteBuffer((unsigned char *)
 buffer.str().c_str(),
                                               buffer.str().length()));
        request.SetCode(code);

        Aws::Lambda::Model::CreateFunctionOutcome outcome =
 client.CreateFunction(
                request);

        if (outcome.IsSuccess()) {
            std::cout << "The lambda function was successfully created. " <<
 seconds
                      << " seconds elapsed." << std::endl;
            break;
        }
        else if (outcome.GetError().GetErrorType() ==
                 Aws::Lambda::LambdaErrors::INVALID_PARAMETER_VALUE &&
                 outcome.GetError().GetMessage().find("role") >= 0) {
            if ((seconds % 5) == 0) { // Log status every 10 seconds.
                std::cout
                        << "Waiting for the IAM role to become available as a
 CreateFunction parameter. "
                        << seconds
                        << " seconds elapsed." << std::endl;

                std::cout << outcome.GetError().GetMessage() << std::endl;
            }
        }
```

```
        else {
            std::cerr << "Error with CreateFunction. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            deleteIamRole(clientConfig);
            return false;
        }
        ++seconds;
        std::this_thread::sleep_for(std::chrono::seconds(1));
    } while (60 > seconds);

    std::cout << "The current Lambda function increments 1 by an input." <<
std::endl;

    // 3.  Invoke the Lambda function.
    {
        int increment = askQuestionForInt("Enter an increment integer: ");

        Aws::Lambda::Model::InvokeResult invokeResult;
        Aws::Utils::Json::JsonValue jsonPayload;
        jsonPayload.WithString("action", "increment");
        jsonPayload.WithInteger("number", increment);
        if (invokeLambdaFunction(jsonPayload, Aws::Lambda::Model::LogType::Tail,
                                 invokeResult, client)) {
            Aws::Utils::Json::JsonValue jsonValue(invokeResult.GetPayload());
            Aws::Map<Aws::String, Aws::Utils::Json::JsonView> values =
                    jsonValue.View().GetAllObjects();
            auto iter = values.find("result");
            if (iter != values.end() && iter->second.IsIntegerType()) {
                {
                    std::cout << INCREMENT_RESUlT_PREFIX
                              << iter->second.AsInteger() << std::endl;
                }
            }
            else {
                std::cout << "There was an error in execution. Here is the log."
                          << std::endl;
                Aws::Utils::ByteBuffer buffer =
Aws::Utils::HashingUtils::Base64Decode(
                        invokeResult.GetLogResult());
                std::cout << "With log " << buffer.GetUnderlyingData() <<
std::endl;
            }
        }
```

```
    }

    std::cout
            << "The Lambda function will now be updated with new code. Press
 return to continue, ";
    Aws::String answer;
    std::getline(std::cin, answer);

    // 4.  Update the Lambda function code.
    {
        Aws::Lambda::Model::UpdateFunctionCodeRequest request;
        request.SetFunctionName(LAMBDA_NAME);
        std::ifstream ifstream(CALCULATOR_LAMBDA_CODE.c_str(),
                               std::ios_base::in | std::ios_base::binary);
        if (!ifstream.is_open()) {
            std::cerr << "Error opening file " << INCREMENT_LAMBDA_CODE << "." <<
 std::endl;

#if USE_CPP_LAMBDA_FUNCTION
            std::cerr
                    << "The cpp Lambda function must be built following the
 instructions in the cpp_lambda/README.md file. "
                    << std::endl;
#endif
            deleteLambdaFunction(client);
            deleteIamRole(clientConfig);
            return false;
        }

        Aws::StringStream buffer;
        buffer << ifstream.rdbuf();
        request.SetZipFile(
                Aws::Utils::ByteBuffer((unsigned char *) buffer.str().c_str(),
                                       buffer.str().length()));
        request.SetPublish(true);

        Aws::Lambda::Model::UpdateFunctionCodeOutcome outcome =
 client.UpdateFunctionCode(
                request);

        if (outcome.IsSuccess()) {
            std::cout << "The lambda code was successfully updated." <<
 std::endl;
        }
```

```
        else {
            std::cerr << "Error with Lambda::UpdateFunctionCode. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
        }
    }

    std::cout
            << "This function uses an environment variable to control the logging
level."
            << std::endl;
    std::cout
            << "UpdateFunctionConfiguration will be used to set the LOG_LEVEL to
DEBUG."
            << std::endl;
    seconds = 0;

    // 5.  Update the Lambda function configuration.
    do {
        ++seconds;
        std::this_thread::sleep_for(std::chrono::seconds(1));
        Aws::Lambda::Model::UpdateFunctionConfigurationRequest request;
        request.SetFunctionName(LAMBDA_NAME);
        Aws::Lambda::Model::Environment environment;
        environment.AddVariables("LOG_LEVEL", "DEBUG");
        request.SetEnvironment(environment);

        Aws::Lambda::Model::UpdateFunctionConfigurationOutcome outcome =
client.UpdateFunctionConfiguration(
                request);

        if (outcome.IsSuccess()) {
            std::cout << "The lambda configuration was successfully updated."
                      << std::endl;
            break;
        }

            // RESOURCE_IN_USE: function code update not completed.
        else if (outcome.GetError().GetErrorType() !=
                 Aws::Lambda::LambdaErrors::RESOURCE_IN_USE) {
            if ((seconds % 10) == 0) { // Log status every 10 seconds.
                std::cout << "Lambda function update in progress . After " <<
seconds
                          << " seconds elapsed." << std::endl;
```

```
            }
        }
        else {
            std::cerr << "Error with Lambda::UpdateFunctionConfiguration. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
        }

    } while (0 < seconds);

    if (0 > seconds) {
        std::cerr << "Function failed to become active." << std::endl;
    }
    else {
        std::cout << "Updated function active after " << seconds << " seconds."
                  << std::endl;
    }

    std::cout
            << "\nThe new code applies an arithmetic operator to two variables, x
 an y."
            << std::endl;
    std::vector<Aws::String> operators = {"plus", "minus", "times", "divided-
by"};
    for (size_t i = 0; i < operators.size(); ++i) {
        std::cout << "   " << i + 1 << " " << operators[i] << std::endl;
    }

    // 6.  Invoke the updated Lambda function.
    do {
        int operatorIndex = askQuestionForIntRange("Select an operator index 1 -
 4 ", 1,
                                                   4);
        int x = askQuestionForInt("Enter an integer for the x value ");
        int y = askQuestionForInt("Enter an integer for the y value ");

        Aws::Utils::Json::JsonValue calculateJsonPayload;
        calculateJsonPayload.WithString("action", operators[operatorIndex - 1]);
        calculateJsonPayload.WithInteger("x", x);
        calculateJsonPayload.WithInteger("y", y);
        Aws::Lambda::Model::InvokeResult calculatedResult;
        if (invokeLambdaFunction(calculateJsonPayload,
                                 Aws::Lambda::Model::LogType::Tail,
                                 calculatedResult, client)) {
```

```
            Aws::Utils::Json::JsonValue jsonValue(calculatedResult.GetPayload());
            Aws::Map<Aws::String, Aws::Utils::Json::JsonView> values =
                    jsonValue.View().GetAllObjects();
            auto iter = values.find("result");
            if (iter != values.end() && iter->second.IsIntegerType()) {
                std::cout << ARITHMETIC_RESULT_PREFIX << x << " "
                          << operators[operatorIndex - 1] << " "
                          << y << " is " << iter->second.AsInteger() <<
std::endl;
            }
            else if (iter != values.end() && iter->second.IsFloatingPointType())
{
                std::cout << ARITHMETIC_RESULT_PREFIX << x << " "
                          << operators[operatorIndex - 1] << " "
                          << y << " is " << iter->second.AsDouble() << std::endl;
            }
            else {
                std::cout << "There was an error in execution. Here is the log."
                          << std::endl;
                Aws::Utils::ByteBuffer buffer =
Aws::Utils::HashingUtils::Base64Decode(
                        calculatedResult.GetLogResult());
                std::cout << "With log " << buffer.GetUnderlyingData() <<
std::endl;
            }
        }

        answer = askQuestion("Would you like to try another operation? (y/n) ");
    } while (answer == "y");

    std::cout
            << "A list of the lambda functions will be retrieved. Press return to
continue, ";
    std::getline(std::cin, answer);

    // 7.  List the Lambda functions.

    std::vector<Aws::String> functions;
    Aws::String marker;

    do {
        Aws::Lambda::Model::ListFunctionsRequest request;
        if (!marker.empty()) {
            request.SetMarker(marker);
```

```
        }

        Aws::Lambda::Model::ListFunctionsOutcome outcome = client.ListFunctions(
                request);

        if (outcome.IsSuccess()) {
            const Aws::Lambda::Model::ListFunctionsResult &result =
outcome.GetResult();
            std::cout << result.GetFunctions().size()
                      << " lambda functions were retrieved." << std::endl;

            for (const Aws::Lambda::Model::FunctionConfiguration
&functionConfiguration: result.GetFunctions()) {
                functions.push_back(functionConfiguration.GetFunctionName());
                std::cout << functions.size() << "  "
                          << functionConfiguration.GetDescription() << std::endl;
                std::cout << "   "
                          <<
Aws::Lambda::Model::RuntimeMapper::GetNameForRuntime(
                                  functionConfiguration.GetRuntime()) << ": "
                          << functionConfiguration.GetHandler()
                          << std::endl;
            }
            marker = result.GetNextMarker();
        }
        else {
            std::cerr << "Error with Lambda::ListFunctions. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
        }
    } while (!marker.empty());

    // 8.  Get a Lambda function.
    if (!functions.empty()) {
        std::stringstream question;
        question << "Choose a function to retrieve between 1 and " <<
functions.size()
                 << " ";
        int functionIndex = askQuestionForIntRange(question.str(), 1,

static_cast<int>(functions.size()));

        Aws::String functionName = functions[functionIndex - 1];
```

```
        Aws::Lambda::Model::GetFunctionRequest request;
        request.SetFunctionName(functionName);

        Aws::Lambda::Model::GetFunctionOutcome outcome =
 client.GetFunction(request);

        if (outcome.IsSuccess()) {
            std::cout << "Function retrieve.\n" <<

 outcome.GetResult().GetConfiguration().Jsonize().View().WriteReadable()
                      << std::endl;
        }
        else {
            std::cerr << "Error with Lambda::GetFunction. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
        }
    }

    std::cout << "The resources will be deleted. Press return to continue, ";
    std::getline(std::cin, answer);

    // 9.  Delete the Lambda function.
    bool result = deleteLambdaFunction(client);

    // 10. Delete the IAM role.
    return result && deleteIamRole(clientConfig);
}

//! Routine which invokes a Lambda function and returns the result.
/*!
 \param jsonPayload: Payload for invoke function.
 \param logType: Log type setting for invoke function.
 \param invokeResult: InvokeResult object to receive the result.
 \param client: Lambda client.
 \return bool: Successful completion.
 */
bool
AwsDoc::Lambda::invokeLambdaFunction(const Aws::Utils::Json::JsonValue
 &jsonPayload,
                                     Aws::Lambda::Model::LogType logType,
                                     Aws::Lambda::Model::InvokeResult
 &invokeResult,
                                     const Aws::Lambda::LambdaClient &client) {
```

```
   int seconds = 0;
   bool result = false;
   /*
    * In this example, the Invoke function can be called before recently created
resources are
    * available.  The Invoke function is called repeatedly until the resources
are
    * available.
    */
   do {
       Aws::Lambda::Model::InvokeRequest request;
       request.SetFunctionName(LAMBDA_NAME);
       request.SetLogType(logType);
       std::shared_ptr<Aws::IOStream> payload =
Aws::MakeShared<Aws::StringStream>(
               "FunctionTest");
       *payload << jsonPayload.View().WriteReadable();
       request.SetBody(payload);
       request.SetContentType("application/json");
       Aws::Lambda::Model::InvokeOutcome outcome = client.Invoke(request);

       if (outcome.IsSuccess()) {
           invokeResult = std::move(outcome.GetResult());
           result = true;
           break;
       }

           // ACCESS_DENIED: because the role is not available yet.
           // RESOURCE_CONFLICT: because the Lambda function is being created or
updated.
       else if ((outcome.GetError().GetErrorType() ==
                 Aws::Lambda::LambdaErrors::ACCESS_DENIED) ||
                (outcome.GetError().GetErrorType() ==
                 Aws::Lambda::LambdaErrors::RESOURCE_CONFLICT)) {
           if ((seconds % 5) == 0) { // Log status every 10 seconds.
               std::cout << "Waiting for the invoke api to be available, status
" <<
                         ((outcome.GetError().GetErrorType() ==
                           Aws::Lambda::LambdaErrors::ACCESS_DENIED ?
                           "ACCESS_DENIED" : "RESOURCE_CONFLICT")) << ". " <<
seconds
                         << " seconds elapsed." << std::endl;
           }
       }
```

```
        else {
            std::cerr << "Error with Lambda::InvokeRequest. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            break;
        }
        ++seconds;
        std::this_thread::sleep_for(std::chrono::seconds(1));
    } while (seconds < 60);

    return result;
}
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK untuk C++ .
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Memanggil
  - ListFunctions
  - UpdateFunctionCode
  - UpdateFunctionConfiguration

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Buat skenario interaktif yang menunjukkan cara memulai fungsi Lambda.

```
import (
 "archive/zip"
 "bytes"
```

```
 "context"
 "encoding/base64"
 "encoding/json"
 "errors"
 "fmt"
 "log"
 "os"
 "strings"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 iamtypes "github.com/aws/aws-sdk-go-v2/service/iam/types"
 "github.com/aws/aws-sdk-go-v2/service/lambda"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/lambda/actions"
)

// GetStartedFunctionsScenario shows you how to use AWS Lambda to perform the
 following
// actions:
//
//  1. Create an AWS Identity and Access Management (IAM) role and Lambda
 function, then upload handler code.
//  2. Invoke the function with a single parameter and get results.
//  3. Update the function code and configure with an environment variable.
//  4. Invoke the function with new parameters and get results. Display the
 returned execution log.
//  5. List the functions for your account, then clean up resources.
type GetStartedFunctionsScenario struct {
 sdkConfig       aws.Config
 functionWrapper actions.FunctionWrapper
 questioner      demotools.IQuestioner
 helper          IScenarioHelper
 isTestRun       bool
}

// NewGetStartedFunctionsScenario constructs a GetStartedFunctionsScenario
 instance from a configuration.
// It uses the specified config to get a Lambda client and create wrappers for
 the actions
// used in the scenario.
func NewGetStartedFunctionsScenario(sdkConfig aws.Config, questioner
 demotools.IQuestioner,
```

```
 helper IScenarioHelper) GetStartedFunctionsScenario {
 lambdaClient := lambda.NewFromConfig(sdkConfig)
 return GetStartedFunctionsScenario{
  sdkConfig:       sdkConfig,
  functionWrapper: actions.FunctionWrapper{LambdaClient: lambdaClient},
  questioner:      questioner,
  helper:          helper,
 }
}

// Run runs the interactive scenario.
func (scenario GetStartedFunctionsScenario) Run(ctx context.Context) {
 defer func() {
  if r := recover(); r != nil {
   log.Printf("Something went wrong with the demo.\n")
  }
 }()

 log.Println(strings.Repeat("-", 88))
 log.Println("Welcome to the AWS Lambda get started with functions demo.")
 log.Println(strings.Repeat("-", 88))

 role := scenario.GetOrCreateRole(ctx)
 funcName := scenario.CreateFunction(ctx, role)
 scenario.InvokeIncrement(ctx, funcName)
 scenario.UpdateFunction(ctx, funcName)
 scenario.InvokeCalculator(ctx, funcName)
 scenario.ListFunctions(ctx)
 scenario.Cleanup(ctx, role, funcName)

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}

// GetOrCreateRole checks whether the specified role exists and returns it if it
 does.
// Otherwise, a role is created that specifies Lambda as a trusted principal.
// The AWSLambdaBasicExecutionRole managed policy is attached to the role and the
 role
// is returned.
func (scenario GetStartedFunctionsScenario) GetOrCreateRole(ctx context.Context)
 *iamtypes.Role {
 var role *iamtypes.Role
```

```
iamClient := iam.NewFromConfig(scenario.sdkConfig)
log.Println("First, we need an IAM role that Lambda can assume.")
roleName := scenario.questioner.Ask("Enter a name for the role:",
demotools.NotEmpty{})
getOutput, err := iamClient.GetRole(ctx, &iam.GetRoleInput{
 RoleName: aws.String(roleName)})
if err != nil {
 var noSuch *iamtypes.NoSuchEntityException
 if errors.As(err, &noSuch) {
  log.Printf("Role %v doesn't exist. Creating it....\n", roleName)
 } else {
  log.Panicf("Couldn't check whether role %v exists. Here's why: %v\n",
   roleName, err)
 }
} else {
 role = getOutput.Role
 log.Printf("Found role %v.\n", *role.RoleName)
}
if role == nil {
 trustPolicy := PolicyDocument{
  Version: "2012-10-17",
  Statement: []PolicyStatement{{
   Effect:    "Allow",
   Principal: map[string]string{"Service": "lambda.amazonaws.com"},
   Action:    []string{"sts:AssumeRole"},
  }},
 }
 policyArn := "arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole"
 createOutput, err := iamClient.CreateRole(ctx, &iam.CreateRoleInput{
  AssumeRolePolicyDocument: aws.String(trustPolicy.String()),
  RoleName:                 aws.String(roleName),
 })
 if err != nil {
  log.Panicf("Couldn't create role %v. Here's why: %v\n", roleName, err)
 }
 role = createOutput.Role
 _, err = iamClient.AttachRolePolicy(ctx, &iam.AttachRolePolicyInput{
  PolicyArn: aws.String(policyArn),
  RoleName:  aws.String(roleName),
 })
 if err != nil {
  log.Panicf("Couldn't attach a policy to role %v. Here's why: %v\n", roleName,
err)
 }
```

```
  log.Printf("Created role %v.\n", *role.RoleName)
  log.Println("Let's give AWS a few seconds to propagate resources...")
  scenario.helper.Pause(10)
 }
 log.Println(strings.Repeat("-", 88))
 return role
}

// CreateFunction creates a Lambda function and uploads a handler written in
 Python.
// The code for the Python handler is packaged as a []byte in .zip format.
func (scenario GetStartedFunctionsScenario) CreateFunction(ctx context.Context,
 role *iamtypes.Role) string {
 log.Println("Let's create a function that increments a number.\n" +
  "The function uses the 'lambda_handler_basic.py' script found in the \n" +
  "'handlers' directory of this project.")
 funcName := scenario.questioner.Ask("Enter a name for the Lambda function:",
 demotools.NotEmpty{})
 zipPackage := scenario.helper.CreateDeploymentPackage("lambda_handler_basic.py",
 fmt.Sprintf("%v.py", funcName))
 log.Printf("Creating function %v and waiting for it to be ready.", funcName)
 funcState := scenario.functionWrapper.CreateFunction(ctx, funcName,
 fmt.Sprintf("%v.lambda_handler", funcName),
  role.Arn, zipPackage)
 log.Printf("Your function is %v.", funcState)
 log.Println(strings.Repeat("-", 88))
 return funcName
}

// InvokeIncrement invokes a Lambda function that increments a number. The
 function
// parameters are contained in a Go struct that is used to serialize the
 parameters to
// a JSON payload that is passed to the function.
// The result payload is deserialized into a Go struct that contains an int
 value.
func (scenario GetStartedFunctionsScenario) InvokeIncrement(ctx context.Context,
 funcName string) {
 parameters := actions.IncrementParameters{Action: "increment"}
 log.Println("Let's invoke our function. This function increments a number.")
 parameters.Number = scenario.questioner.AskInt("Enter a number to increment:",
 demotools.NotEmpty{})
 log.Printf("Invoking %v with %v...\n", funcName, parameters.Number)
```

```
 invokeOutput := scenario.functionWrapper.Invoke(ctx, funcName, parameters,
 false)
 var payload actions.LambdaResultInt
 err := json.Unmarshal(invokeOutput.Payload, &payload)
 if err != nil {
  log.Panicf("Couldn't unmarshal payload from invoking %v. Here's why: %v\n",
   funcName, err)
 }
 log.Printf("Invoking %v with %v returned %v.\n", funcName, parameters.Number,
 payload)
 log.Println(strings.Repeat("-", 88))
}

// UpdateFunction updates the code for a Lambda function by uploading a simple
 arithmetic
// calculator written in Python. The code for the Python handler is packaged as a
// []byte in .zip format.
// After the code is updated, the configuration is also updated with a new log
// level that instructs the handler to log additional information.
func (scenario GetStartedFunctionsScenario) UpdateFunction(ctx context.Context,
 funcName string) {
 log.Println("Let's update the function to an arithmetic calculator.\n" +
  "The function uses the 'lambda_handler_calculator.py' script found in the \n" +
  "'handlers' directory of this project.")
 scenario.questioner.Ask("Press Enter when you're ready.")
 log.Println("Creating deployment package...")
 zipPackage :=
 scenario.helper.CreateDeploymentPackage("lambda_handler_calculator.py",
  fmt.Sprintf("%v.py", funcName))
 log.Println("...and updating the Lambda function and waiting for it to be
 ready.")
 funcState := scenario.functionWrapper.UpdateFunctionCode(ctx, funcName,
 zipPackage)
 log.Printf("Updated function %v. Its current state is %v.", funcName, funcState)
 log.Println("This function uses an environment variable to control logging
 level.")
 log.Println("Let's set it to DEBUG to get the most logging.")
 scenario.functionWrapper.UpdateFunctionConfiguration(ctx, funcName,
  map[string]string{"LOG_LEVEL": "DEBUG"})
 log.Println(strings.Repeat("-", 88))
}

// InvokeCalculator invokes the Lambda calculator function. The parameters are
 stored in a
```

```
// Go struct that is used to serialize the parameters to a JSON payload. That
 payload is then passed
// to the function.
// The result payload is deserialized to a Go struct that stores the result as
 either an
// int or float32, depending on the kind of operation that was specified.
func (scenario GetStartedFunctionsScenario) InvokeCalculator(ctx context.Context,
 funcName string) {
 wantInvoke := true
 choices := []string{"plus", "minus", "times", "divided-by"}
 for wantInvoke {
  choice := scenario.questioner.AskChoice("Select an arithmetic operation:\n",
 choices)
  x := scenario.questioner.AskInt("Enter a value for x:", demotools.NotEmpty{})
  y := scenario.questioner.AskInt("Enter a value for y:", demotools.NotEmpty{})
  log.Printf("Invoking %v %v %v...", x, choices[choice], y)
  calcParameters := actions.CalculatorParameters{
   Action: choices[choice],
   X:      x,
   Y:      y,
  }
  invokeOutput := scenario.functionWrapper.Invoke(ctx, funcName, calcParameters,
 true)
  var payload any
  if choice == 3 { // divide-by results in a float.
   payload = actions.LambdaResultFloat{}
  } else {
   payload = actions.LambdaResultInt{}
  }
  err := json.Unmarshal(invokeOutput.Payload, &payload)
  if err != nil {
   log.Panicf("Couldn't unmarshal payload from invoking %v. Here's why: %v\n",
    funcName, err)
  }
  log.Printf("Invoking %v with %v %v %v returned %v.\n", funcName,
   calcParameters.X, calcParameters.Action, calcParameters.Y, payload)
  scenario.questioner.Ask("Press Enter to see the logs from the call.")
  logRes, err := base64.StdEncoding.DecodeString(*invokeOutput.LogResult)
  if err != nil {
   log.Panicf("Couldn't decode log result. Here's why: %v\n", err)
  }
  log.Println(string(logRes))
  wantInvoke = scenario.questioner.AskBool("Do you want to calculate again? (y/
n)", "y")
```

```
 }
 log.Println(strings.Repeat("-", 88))
}

// ListFunctions lists up to the specified number of functions for your account.
func (scenario GetStartedFunctionsScenario) ListFunctions(ctx context.Context) {
 count := scenario.questioner.AskInt(
  "Let's list functions for your account. How many do you want to see?",
 demotools.NotEmpty{})
 functions := scenario.functionWrapper.ListFunctions(ctx, count)
 log.Printf("Found %v functions:", len(functions))
 for _, function := range functions {
  log.Printf("\t%v", *function.FunctionName)
 }
 log.Println(strings.Repeat("-", 88))
}

// Cleanup removes the IAM and Lambda resources created by the example.
func (scenario GetStartedFunctionsScenario) Cleanup(ctx context.Context, role
 *iamtypes.Role, funcName string) {
 if scenario.questioner.AskBool("Do you want to clean up resources created for
 this example? (y/n)",
  "y") {
  iamClient := iam.NewFromConfig(scenario.sdkConfig)
  policiesOutput, err := iamClient.ListAttachedRolePolicies(ctx,
   &iam.ListAttachedRolePoliciesInput{RoleName: role.RoleName})
  if err != nil {
   log.Panicf("Couldn't get policies attached to role %v. Here's why: %v\n",
    *role.RoleName, err)
  }
  for _, policy := range policiesOutput.AttachedPolicies {
   _, err = iamClient.DetachRolePolicy(ctx, &iam.DetachRolePolicyInput{
    PolicyArn: policy.PolicyArn, RoleName: role.RoleName,
   })
   if err != nil {
    log.Panicf("Couldn't detach policy %v from role %v. Here's why: %v\n",
     *policy.PolicyArn, *role.RoleName, err)
   }
  }
  _, err = iamClient.DeleteRole(ctx, &iam.DeleteRoleInput{RoleName:
 role.RoleName})
  if err != nil {
   log.Panicf("Couldn't delete role %v. Here's why: %v\n", *role.RoleName, err)
  }
```

```
  log.Printf("Deleted role %v.\n", *role.RoleName)

  scenario.functionWrapper.DeleteFunction(ctx, funcName)
  log.Printf("Deleted function %v.\n", funcName)
 } else {
  log.Println("Okay. Don't forget to delete the resources when you're done with
 them.")
 }
}

// IScenarioHelper abstracts I/O and wait functions from a scenario so that they
// can be mocked for unit testing.
type IScenarioHelper interface {
 Pause(secs int)
 CreateDeploymentPackage(sourceFile string, destinationFile string) *bytes.Buffer
}

// ScenarioHelper lets the caller specify the path to Lambda handler functions.
type ScenarioHelper struct {
 HandlerPath string
}

// Pause waits for the specified number of seconds.
func (helper *ScenarioHelper) Pause(secs int) {
 time.Sleep(time.Duration(secs) * time.Second)
}

// CreateDeploymentPackage creates an AWS Lambda deployment package from a source
 file. The
// deployment package is stored in .zip format in a bytes.Buffer. The buffer can
 be
// used to pass a []byte to Lambda when creating the function.
// The specified destinationFile is the name to give the file when it's deployed
 to Lambda.
func (helper *ScenarioHelper) CreateDeploymentPackage(sourceFile string,
 destinationFile string) *bytes.Buffer {
 var err error
 buffer := &bytes.Buffer{}
 writer := zip.NewWriter(buffer)
 zFile, err := writer.Create(destinationFile)
 if err != nil {
  log.Panicf("Couldn't create destination archive %v. Here's why: %v\n",
 destinationFile, err)
 }
```

```
 sourceBody, err := os.ReadFile(fmt.Sprintf("%v/%v", helper.HandlerPath,
 sourceFile))
 if err != nil {
  log.Panicf("Couldn't read handler source file %v. Here's why: %v\n",
   sourceFile, err)
 } else {
  _, err = zFile.Write(sourceBody)
  if err != nil {
   log.Panicf("Couldn't write handler %v to zip archive. Here's why: %v\n",
    sourceFile, err)
  }
 }
 err = writer.Close()
 if err != nil {
  log.Panicf("Couldn't close zip writer. Here's why: %v\n", err)
 }
 return buffer
}
```

Buat struct yang membungkus tindakan Lambda individual.

```
import (
 "bytes"
 "context"
 "encoding/json"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/lambda"
 "github.com/aws/aws-sdk-go-v2/service/lambda/types"
)

// FunctionWrapper encapsulates function actions used in the examples.
// It contains an AWS Lambda service client that is used to perform user actions.
type FunctionWrapper struct {
 LambdaClient *lambda.Client
}
```

```
// GetFunction gets data about the Lambda function specified by functionName.
func (wrapper FunctionWrapper) GetFunction(ctx context.Context, functionName
 string) types.State {
 var state types.State
 funcOutput, err := wrapper.LambdaClient.GetFunction(ctx,
 &lambda.GetFunctionInput{
  FunctionName: aws.String(functionName),
 })
 if err != nil {
  log.Panicf("Couldn't get function %v. Here's why: %v\n", functionName, err)
 } else {
  state = funcOutput.Configuration.State
 }
 return state
}



// CreateFunction creates a new Lambda function from code contained in the
 zipPackage
// buffer. The specified handlerName must match the name of the file and function
// contained in the uploaded code. The role specified by iamRoleArn is assumed by
// Lambda and grants specific permissions.
// When the function already exists, types.StateActive is returned.
// When the function is created, a lambda.FunctionActiveV2Waiter is used to wait
 until the
// function is active.
func (wrapper FunctionWrapper) CreateFunction(ctx context.Context, functionName
 string, handlerName string,
 iamRoleArn *string, zipPackage *bytes.Buffer) types.State {
 var state types.State
 _, err := wrapper.LambdaClient.CreateFunction(ctx, &lambda.CreateFunctionInput{
  Code:         &types.FunctionCode{ZipFile: zipPackage.Bytes()},
  FunctionName: aws.String(functionName),
  Role:         iamRoleArn,
  Handler:      aws.String(handlerName),
  Publish:      true,
  Runtime:      types.RuntimePython39,
 })
 if err != nil {
  var resConflict *types.ResourceConflictException
  if errors.As(err, &resConflict) {
   log.Printf("Function %v already exists.\n", functionName)
```

```
    state = types.StateActive
   } else {
    log.Panicf("Couldn't create function %v. Here's why: %v\n", functionName, err)
   }
  } else {
   waiter := lambda.NewFunctionActiveV2Waiter(wrapper.LambdaClient)
   funcOutput, err := waiter.WaitForOutput(ctx, &lambda.GetFunctionInput{
    FunctionName: aws.String(functionName)}, 1*time.Minute)
   if err != nil {
    log.Panicf("Couldn't wait for function %v to be active. Here's why: %v\n",
  functionName, err)
   } else {
    state = funcOutput.Configuration.State
   }
  }
  return state
}



// UpdateFunctionCode updates the code for the Lambda function specified by
  functionName.
// The existing code for the Lambda function is entirely replaced by the code in
  the
// zipPackage buffer. After the update action is called, a
  lambda.FunctionUpdatedV2Waiter
// is used to wait until the update is successful.
func (wrapper FunctionWrapper) UpdateFunctionCode(ctx context.Context,
  functionName string, zipPackage *bytes.Buffer) types.State {
  var state types.State
  _, err := wrapper.LambdaClient.UpdateFunctionCode(ctx,
  &lambda.UpdateFunctionCodeInput{
   FunctionName: aws.String(functionName), ZipFile: zipPackage.Bytes(),
  })
  if err != nil {
   log.Panicf("Couldn't update code for function %v. Here's why: %v\n",
  functionName, err)
  } else {
   waiter := lambda.NewFunctionUpdatedV2Waiter(wrapper.LambdaClient)
   funcOutput, err := waiter.WaitForOutput(ctx, &lambda.GetFunctionInput{
    FunctionName: aws.String(functionName)}, 1*time.Minute)
   if err != nil {
    log.Panicf("Couldn't wait for function %v to be active. Here's why: %v\n",
  functionName, err)
```

```
  } else {
   state = funcOutput.Configuration.State
  }
 }
 return state
}



// UpdateFunctionConfiguration updates a map of environment variables configured
 for
// the Lambda function specified by functionName.
func (wrapper FunctionWrapper) UpdateFunctionConfiguration(ctx context.Context,
 functionName string, envVars map[string]string) {
 _, err := wrapper.LambdaClient.UpdateFunctionConfiguration(ctx,
 &lambda.UpdateFunctionConfigurationInput{
  FunctionName: aws.String(functionName),
  Environment:  &types.Environment{Variables: envVars},
 })
 if err != nil {
  log.Panicf("Couldn't update configuration for %v. Here's why: %v",
 functionName, err)
 }
}



// ListFunctions lists up to maxItems functions for the account. This function
 uses a
// lambda.ListFunctionsPaginator to paginate the results.
func (wrapper FunctionWrapper) ListFunctions(ctx context.Context, maxItems int)
 []types.FunctionConfiguration {
 var functions []types.FunctionConfiguration
 paginator := lambda.NewListFunctionsPaginator(wrapper.LambdaClient,
 &lambda.ListFunctionsInput{
  MaxItems: aws.Int32(int32(maxItems)),
 })
 for paginator.HasMorePages() && len(functions) < maxItems {
  pageOutput, err := paginator.NextPage(ctx)
  if err != nil {
   log.Panicf("Couldn't list functions for your account. Here's why: %v\n", err)
  }
  functions = append(functions, pageOutput.Functions...)
 }
```

```
 return functions
}



// DeleteFunction deletes the Lambda function specified by functionName.
func (wrapper FunctionWrapper) DeleteFunction(ctx context.Context, functionName
 string) {
 _, err := wrapper.LambdaClient.DeleteFunction(ctx, &lambda.DeleteFunctionInput{
  FunctionName: aws.String(functionName),
 })
 if err != nil {
  log.Panicf("Couldn't delete function %v. Here's why: %v\n", functionName, err)
 }
}



// Invoke invokes the Lambda function specified by functionName, passing the
 parameters
// as a JSON payload. When getLog is true, types.LogTypeTail is specified, which
 tells
// Lambda to include the last few log lines in the returned result.
func (wrapper FunctionWrapper) Invoke(ctx context.Context, functionName string,
 parameters any, getLog bool) *lambda.InvokeOutput {
 logType := types.LogTypeNone
 if getLog {
  logType = types.LogTypeTail
 }
 payload, err := json.Marshal(parameters)
 if err != nil {
  log.Panicf("Couldn't marshal parameters to JSON. Here's why %v\n", err)
 }
 invokeOutput, err := wrapper.LambdaClient.Invoke(ctx, &lambda.InvokeInput{
  FunctionName: aws.String(functionName),
  LogType:      logType,
  Payload:      payload,
 })
 if err != nil {
  log.Panicf("Couldn't invoke function %v. Here's why: %v\n", functionName, err)
 }
 return invokeOutput
}
```

```
// IncrementParameters is used to serialize parameters to the increment Lambda
 handler.
type IncrementParameters struct {
 Action string `json:"action"`
 Number int    `json:"number"`
}

// CalculatorParameters is used to serialize parameters to the calculator Lambda
 handler.
type CalculatorParameters struct {
 Action string `json:"action"`
 X      int    `json:"x"`
 Y      int    `json:"y"`
}

// LambdaResultInt is used to deserialize an int result from a Lambda handler.
type LambdaResultInt struct {
 Result int `json:"result"`
}

// LambdaResultFloat is used to deserialize a float32 result from a Lambda
 handler.
type LambdaResultFloat struct {
 Result float32 `json:"result"`
}
```

Tentukan handler Lambda yang menambah angka.

```
import logging

logger = logging.getLogger()
logger.setLevel(logging.INFO)


def lambda_handler(event, context):
    """
    Accepts an action and a single number, performs the specified action on the
 number,
    and returns the result. The only allowable action is 'increment'.
```

```
    :param event: The event dict that contains the parameters sent when the
 function
                  is invoked.
    :param context: The context in which the function is called.
    :return: The result of the action.
    """
    result = None
    action = event.get("action")
    if action == "increment":
        result = event.get("number", 0) + 1
        logger.info("Calculated result of %s", result)
    else:
        logger.error("%s is not a valid action.", action)

    response = {"result": result}
    return response
```

Tentukan handler Lambda kedua yang melakukan operasi aritmatika.

```
import logging
import os


logger = logging.getLogger()

# Define a list of Python lambda functions that are called by this AWS Lambda
 function.
ACTIONS = {
    "plus": lambda x, y: x + y,
    "minus": lambda x, y: x - y,
    "times": lambda x, y: x * y,
    "divided-by": lambda x, y: x / y,
}


def lambda_handler(event, context):
    """
    Accepts an action and two numbers, performs the specified action on the
 numbers,
```

```
    and returns the result.

    :param event: The event dict that contains the parameters sent when the
function
                  is invoked.
    :param context: The context in which the function is called.
    :return: The result of the specified action.
    """
    # Set the log level based on a variable configured in the Lambda environment.
    logger.setLevel(os.environ.get("LOG_LEVEL", logging.INFO))
    logger.debug("Event: %s", event)

    action = event.get("action")
    func = ACTIONS.get(action)
    x = event.get("x")
    y = event.get("y")
    result = None
    try:
        if func is not None and x is not None and y is not None:
            result = func(x, y)
            logger.info("%s %s %s is %s", x, action, y, result)
        else:
            logger.error("I can't calculate %s %s %s.", x, action, y)
    except ZeroDivisionError:
        logger.warning("I can't divide %s by 0!", x)

    response = {"result": result}
    return response
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK untuk Go .
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Memanggil
  - ListFunctions
  - UpdateFunctionCode
  - UpdateFunctionConfiguration

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkap dan pelajari cara menyiapkan dan menjalankan di Repositori Contoh Kode AWS.

```
/*
 *  Lambda function names appear as:
 *
 *  arn:aws:lambda:us-west-2:335556666777:function:HelloFunction
 *
 *  To find this value, look at the function in the AWS Management Console.
 *
 *  Before running this Java code example, set up your development environment,
 including your credentials.
 *
 *  For more information, see this documentation topic:
 *
 *  https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 *  This example performs the following tasks:
 *
 * 1. Creates an AWS Lambda function.
 * 2. Gets a specific AWS Lambda function.
 * 3. Lists all Lambda functions.
 * 4. Invokes a Lambda function.
 * 5. Updates the Lambda function code and invokes it again.
 * 6. Updates a Lambda function's configuration value.
 * 7. Deletes a Lambda function.
 */

public class LambdaScenario {
    public static final String DASHES = new String(new char[80]).replace("\0",
 "-");

    public static void main(String[] args) throws InterruptedException {
        final String usage = """
```

```
            Usage:
                <functionName> <role> <handler> <bucketName> <key>\s

            Where:
                functionName - The name of the Lambda function.\s
                role - The AWS Identity and Access Management (IAM) service role
that has Lambda permissions.\s
                handler - The fully qualified method name (for example,
example.Handler::handleRequest).\s
                bucketName - The Amazon Simple Storage Service (Amazon S3) bucket
name that contains the .zip or .jar used to update the Lambda function's code.\s
                key - The Amazon S3 key name that represents the .zip or .jar
(for example, LambdaHello-1.0-SNAPSHOT.jar).
                """;

        if (args.length != 5) {
              System.out.println(usage);
              return;
        }

        String functionName = args[0];
        String role = args[1];
        String handler = args[2];
        String bucketName = args[3];
        String key = args[4];
        LambdaClient awsLambda = LambdaClient.builder()
            .build();

        System.out.println(DASHES);
        System.out.println("Welcome to the AWS Lambda Basics scenario.");
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("1. Create an AWS Lambda function.");
        String funArn = createLambdaFunction(awsLambda, functionName, key,
bucketName, role, handler);
        System.out.println("The AWS Lambda ARN is " + funArn);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("2. Get the " + functionName + " AWS Lambda
function.");
        getFunction(awsLambda, functionName);
```

```
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("3. List all AWS Lambda functions.");
        listFunctions(awsLambda);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("4. Invoke the Lambda function.");
        System.out.println("*** Sleep for 1 min to get Lambda function ready.");
        Thread.sleep(60000);
        invokeFunction(awsLambda, functionName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("5. Update the Lambda function code and invoke it
again.");
        updateFunctionCode(awsLambda, functionName, bucketName, key);
        System.out.println("*** Sleep for 1 min to get Lambda function ready.");
        Thread.sleep(60000);
        invokeFunction(awsLambda, functionName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("6. Update a Lambda function's configuration value.");
        updateFunctionConfiguration(awsLambda, functionName, handler);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("7. Delete the AWS Lambda function.");
        LambdaScenario.deleteLambdaFunction(awsLambda, functionName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("The AWS Lambda scenario completed successfully");
        System.out.println(DASHES);
        awsLambda.close();
    }

    /**
     * Creates a new Lambda function in AWS using the AWS Lambda Java API.
     *
     * @param awsLambda    the AWS Lambda client used to interact with the AWS
Lambda service
```

```
     * @param functionName the name of the Lambda function to create
     * @param key          the S3 key of the function code
     * @param bucketName   the name of the S3 bucket containing the function code
     * @param role         the IAM role to assign to the Lambda function
     * @param handler      the fully qualified class name of the function handler
     * @return the Amazon Resource Name (ARN) of the created Lambda function
     */
    public static String createLambdaFunction(LambdaClient awsLambda,
                                               String functionName,
                                               String key,
                                               String bucketName,
                                               String role,
                                               String handler) {

        try {
            LambdaWaiter waiter = awsLambda.waiter();
            FunctionCode code = FunctionCode.builder()
                .s3Key(key)
                .s3Bucket(bucketName)
                .build();

            CreateFunctionRequest functionRequest =
CreateFunctionRequest.builder()
                .functionName(functionName)
                .description("Created by the Lambda Java API")
                .code(code)
                .handler(handler)
                .runtime(Runtime.JAVA17)
                .role(role)
                .build();

            // Create a Lambda function using a waiter
            CreateFunctionResponse functionResponse =
awsLambda.createFunction(functionRequest);
            GetFunctionRequest getFunctionRequest = GetFunctionRequest.builder()
                .functionName(functionName)
                .build();
            WaiterResponse<GetFunctionResponse> waiterResponse =
waiter.waitUntilFunctionExists(getFunctionRequest);
            waiterResponse.matched().response().ifPresent(System.out::println);
            return functionResponse.functionArn();

        } catch (LambdaException e) {
            System.err.println(e.getMessage());
```

```
            System.exit(1);
        }
        return "";
    }

    /**
     * Retrieves information about an AWS Lambda function.
     *
     * @param awsLambda    an instance of the {@link LambdaClient} class, which
is used to interact with the AWS Lambda service
     * @param functionName the name of the AWS Lambda function to retrieve
information about
     */
    public static void getFunction(LambdaClient awsLambda, String functionName) {
        try {
            GetFunctionRequest functionRequest = GetFunctionRequest.builder()
                .functionName(functionName)
                .build();

            GetFunctionResponse response =
awsLambda.getFunction(functionRequest);
            System.out.println("The runtime of this Lambda function is " +
response.configuration().runtime());

        } catch (LambdaException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    /**
     * Lists the AWS Lambda functions associated with the current AWS account.
     *
     * @param awsLambda an instance of the {@link LambdaClient} class, which is
used to interact with the AWS Lambda service
     *
     * @throws LambdaException if an error occurs while interacting with the AWS
Lambda service
     */
    public static void listFunctions(LambdaClient awsLambda) {
        try {
            ListFunctionsResponse functionResult = awsLambda.listFunctions();
            List<FunctionConfiguration> list = functionResult.functions();
            for (FunctionConfiguration config : list) {
```

```
                System.out.println("The function name is " +
config.functionName());
            }

        } catch (LambdaException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    /**
     * Invokes a specific AWS Lambda function.
     *
     * @param awsLambda    an instance of {@link LambdaClient} to interact with
the AWS Lambda service
     * @param functionName the name of the AWS Lambda function to be invoked
     */
    public static void invokeFunction(LambdaClient awsLambda, String
functionName) {
        InvokeResponse res;
        try {
            // Need a SdkBytes instance for the payload.
            JSONObject jsonObj = new JSONObject();
            jsonObj.put("inputValue", "2000");
            String json = jsonObj.toString();
            SdkBytes payload = SdkBytes.fromUtf8String(json);

            InvokeRequest request = InvokeRequest.builder()
                .functionName(functionName)
                .payload(payload)
                .build();

            res = awsLambda.invoke(request);
            String value = res.payload().asUtf8String();
            System.out.println(value);

        } catch (LambdaException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    /**
     * Updates the code for an AWS Lambda function.
```

```
    *
    * @param awsLambda  the AWS Lambda client
    * @param functionName the name of the Lambda function to update
    * @param bucketName the name of the S3 bucket where the function code is
located
    * @param key the key (file name) of the function code in the S3 bucket
    * @throws LambdaException if there is an error updating the function code
    */
   public static void updateFunctionCode(LambdaClient awsLambda, String
functionName, String bucketName, String key) {
       try {
           LambdaWaiter waiter = awsLambda.waiter();
           UpdateFunctionCodeRequest functionCodeRequest =
UpdateFunctionCodeRequest.builder()
                .functionName(functionName)
                .publish(true)
                .s3Bucket(bucketName)
                .s3Key(key)
                .build();

           UpdateFunctionCodeResponse response =
awsLambda.updateFunctionCode(functionCodeRequest);
           GetFunctionConfigurationRequest getFunctionConfigRequest =
GetFunctionConfigurationRequest.builder()
                .functionName(functionName)
                .build();

           WaiterResponse<GetFunctionConfigurationResponse> waiterResponse =
waiter
                .waitUntilFunctionUpdated(getFunctionConfigRequest);
           waiterResponse.matched().response().ifPresent(System.out::println);
           System.out.println("The last modified value is " +
response.lastModified());

       } catch (LambdaException e) {
           System.err.println(e.getMessage());
           System.exit(1);
       }
   }

   /**
    * Updates the configuration of an AWS Lambda function.
    *
```

```
    * @param awsLambda     the {@link LambdaClient} instance to use for the AWS
Lambda operation
    * @param functionName  the name of the AWS Lambda function to update
    * @param handler       the new handler for the AWS Lambda function
    *
    * @throws LambdaException if there is an error while updating the function
configuration
    */
   public static void updateFunctionConfiguration(LambdaClient awsLambda, String
functionName, String handler) {
       try {
           UpdateFunctionConfigurationRequest configurationRequest =
UpdateFunctionConfigurationRequest.builder()
               .functionName(functionName)
               .handler(handler)
               .runtime(Runtime.JAVA17)
               .build();

           awsLambda.updateFunctionConfiguration(configurationRequest);

       } catch (LambdaException e) {
           System.err.println(e.getMessage());
           System.exit(1);
       }
   }

   /**
    * Deletes an AWS Lambda function.
    *
    * @param awsLambda     an instance of the {@link LambdaClient} class, which
is used to interact with the AWS Lambda service
    * @param functionName  the name of the Lambda function to be deleted
    *
    * @throws LambdaException if an error occurs while deleting the Lambda
function
    */
   public static void deleteLambdaFunction(LambdaClient awsLambda, String
functionName) {
       try {
           DeleteFunctionRequest request = DeleteFunctionRequest.builder()
               .functionName(functionName)
               .build();

           awsLambda.deleteFunction(request);
```

```
            System.out.println("The " + functionName + " function was deleted");

        } catch (LambdaException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK for Java 2.x .
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Memanggil
  - ListFunctions
  - UpdateFunctionCode
  - UpdateFunctionConfiguration

JavaScript

SDK untuk JavaScript (V3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Buat AWS Identity and Access Management Peran (IAM) yang memberikan izin Lambda untuk menulis ke log.

```
    logger.log(`Creating role (${NAME_ROLE_LAMBDA})...`);
    const response = await createRole(NAME_ROLE_LAMBDA);

import { AttachRolePolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});
```

```
/**
 *
 * @param {string} policyArn
 * @param {string} roleName
 */
export const attachRolePolicy = (policyArn, roleName) => {
  const command = new AttachRolePolicyCommand({
    PolicyArn: policyArn,
    RoleName: roleName,
  });

  return client.send(command);
};
```

Buat fungsi Lambda dan unggah kode handler.

```
const createFunction = async (funcName, roleArn) => {
  const client = new LambdaClient({});
  const code = await readFile(`${dirname}../functions/${funcName}.zip`);

  const command = new CreateFunctionCommand({
    Code: { ZipFile: code },
    FunctionName: funcName,
    Role: roleArn,
    Architectures: [Architecture.arm64],
    Handler: "index.handler", // Required when sending a .zip file
    PackageType: PackageType.Zip, // Required when sending a .zip file
    Runtime: Runtime.nodejs16x, // Required when sending a .zip file
  });

  return client.send(command);
};
```

Memanggil fungsi dengan satu parameter dan dapatkan hasil.

```
const invoke = async (funcName, payload) => {
  const client = new LambdaClient({});
  const command = new InvokeCommand({
    FunctionName: funcName,
    Payload: JSON.stringify(payload),
```

```
    LogType: LogType.Tail,
  });

  const { Payload, LogResult } = await client.send(command);
  const result = Buffer.from(Payload).toString();
  const logs = Buffer.from(LogResult, "base64").toString();
  return { logs, result };
};
```

Perbarui kode fungsi dan konfigurasikan lingkungan Lambda dengan variabel lingkungan.

```
const updateFunctionCode = async (funcName, newFunc) => {
  const client = new LambdaClient({});
  const code = await readFile(`${dirname}../functions/${newFunc}.zip`);
  const command = new UpdateFunctionCodeCommand({
    ZipFile: code,
    FunctionName: funcName,
    Architectures: [Architecture.arm64],
    Handler: "index.handler", // Required when sending a .zip file
    PackageType: PackageType.Zip, // Required when sending a .zip file
    Runtime: Runtime.nodejs16x, // Required when sending a .zip file
  });

  return client.send(command);
};

const updateFunctionConfiguration = async (funcName) => {
  const client = new LambdaClient({});
  const config = readFileSync(`${dirname}../functions/config.json`).toString();
  const command = new UpdateFunctionConfigurationCommand({
    ...JSON.parse(config),
    FunctionName: funcName,
  });
  const result = await client.send(command);
  await waitForFunctionUpdated({ FunctionName: funcName });
  return result;
};
```

Buat daftar fungsi untuk akun Anda.

```
const listFunctions = () => {
```

```
  const client = new LambdaClient({});
  const command = new ListFunctionsCommand({});

  return client.send(command);
};
```

Hapus peran IAM dan fungsi Lambda.

```
import { DeleteRoleCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} roleName
 */
export const deleteRole = (roleName) => {
  const command = new DeleteRoleCommand({ RoleName: roleName });
  return client.send(command);
};

/**
 * @param {string} funcName
 */
const deleteFunction = (funcName) => {
  const client = new LambdaClient({});
  const command = new DeleteFunctionCommand({ FunctionName: funcName });
  return client.send(command);
};
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK untuk JavaScript .
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Memanggil
  - ListFunctions
  - UpdateFunctionCode
  - UpdateFunctionConfiguration

Kotlin

SDK untuk Kotlin

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkap dan pelajari cara menyiapkan dan menjalankan di Repositori Contoh Kode AWS.

```
suspend fun main(args: Array<String>) {
    val usage = """
        Usage:
            <functionName> <role> <handler> <bucketName> <updatedBucketName>
 <key>

        Where:
            functionName - The name of the AWS Lambda function.
            role - The AWS Identity and Access Management (IAM) service role that
 has AWS Lambda permissions.
            handler - The fully qualified method name (for example,
 example.Handler::handleRequest).
            bucketName - The Amazon Simple Storage Service (Amazon S3) bucket
 name that contains the ZIP or JAR used for the Lambda function's code.
            updatedBucketName - The Amazon S3 bucket name that contains the .zip
 or .jar used to update the Lambda function's code.
            key - The Amazon S3 key name that represents the .zip or .jar file
 (for example, LambdaHello-1.0-SNAPSHOT.jar).
            """

    if (args.size != 6) {
        println(usage)
        exitProcess(1)
    }

    val functionName = args[0]
    val role = args[1]
    val handler = args[2]
    val bucketName = args[3]
    val updatedBucketName = args[4]
    val key = args[5]
```

```
    println("Creating a Lambda function named $functionName.")
    val funArn = createScFunction(functionName, bucketName, key, handler, role)
    println("The AWS Lambda ARN is $funArn")

    // Get a specific Lambda function.
    println("Getting the $functionName AWS Lambda function.")
    getFunction(functionName)

    // List the Lambda functions.
    println("Listing all AWS Lambda functions.")
    listFunctionsSc()

    // Invoke the Lambda function.
    println("*** Invoke the Lambda function.")
    invokeFunctionSc(functionName)

    // Update the AWS Lambda function code.
    println("*** Update the Lambda function code.")
    updateFunctionCode(functionName, updatedBucketName, key)

    // println("*** Invoke the function again after updating the code.")
    invokeFunctionSc(functionName)

    // Update the AWS Lambda function configuration.
    println("Update the run time of the function.")
    updateFunctionConfiguration(functionName, handler)

    // Delete the AWS Lambda function.
    println("Delete the AWS Lambda function.")
    delFunction(functionName)
}

suspend fun createScFunction(
    myFunctionName: String,
    s3BucketName: String,
    myS3Key: String,
    myHandler: String,
    myRole: String,
): String {
    val functionCode =
        FunctionCode {
            s3Bucket = s3BucketName
            s3Key = myS3Key
        }
```

```
    val request =
        CreateFunctionRequest {
            functionName = myFunctionName
            code = functionCode
            description = "Created by the Lambda Kotlin API"
            handler = myHandler
            role = myRole
            runtime = Runtime.Java17
        }

    // Create a Lambda function using a waiter
    LambdaClient { region = "us-east-1" }.use { awsLambda ->
        val functionResponse = awsLambda.createFunction(request)
        awsLambda.waitUntilFunctionActive {
            functionName = myFunctionName
        }
        return functionResponse.functionArn.toString()
    }
}

suspend fun getFunction(functionNameVal: String) {
    val functionRequest =
        GetFunctionRequest {
            functionName = functionNameVal
        }

    LambdaClient { region = "us-east-1" }.use { awsLambda ->
        val response = awsLambda.getFunction(functionRequest)
        println("The runtime of this Lambda function is
 ${response.configuration?.runtime}")
    }
}

suspend fun listFunctionsSc() {
    val request =
        ListFunctionsRequest {
            maxItems = 10
        }

    LambdaClient { region = "us-east-1" }.use { awsLambda ->
        val response = awsLambda.listFunctions(request)
        response.functions?.forEach { function ->
            println("The function name is ${function.functionName}")
```

```
        }
    }
}

suspend fun invokeFunctionSc(functionNameVal: String) {
    val json = """{"inputValue":"1000"}"""
    val byteArray = json.trimIndent().encodeToByteArray()
    val request =
        InvokeRequest {
            functionName = functionNameVal
            payload = byteArray
            logType = LogType.Tail
        }

    LambdaClient { region = "us-east-1" }.use { awsLambda ->
        val res = awsLambda.invoke(request)
        println("The function payload is
 ${res.payload?.toString(Charsets.UTF_8)}")
    }
}

suspend fun updateFunctionCode(
    functionNameVal: String?,
    bucketName: String?,
    key: String?,
) {
    val functionCodeRequest =
        UpdateFunctionCodeRequest {
            functionName = functionNameVal
            publish = true
            s3Bucket = bucketName
            s3Key = key
        }

    LambdaClient { region = "us-east-1" }.use { awsLambda ->
        val response = awsLambda.updateFunctionCode(functionCodeRequest)
        awsLambda.waitUntilFunctionUpdated {
            functionName = functionNameVal
        }
        println("The last modified value is " + response.lastModified)
    }
}

suspend fun updateFunctionConfiguration(
```

```
    functionNameVal: String?,
    handlerVal: String?,
) {
    val configurationRequest =
        UpdateFunctionConfigurationRequest {
            functionName = functionNameVal
            handler = handlerVal
            runtime = Runtime.Java17
        }

    LambdaClient { region = "us-east-1" }.use { awsLambda ->
        awsLambda.updateFunctionConfiguration(configurationRequest)
    }
}

suspend fun delFunction(myFunctionName: String) {
    val request =
        DeleteFunctionRequest {
            functionName = myFunctionName
        }

    LambdaClient { region = "us-east-1" }.use { awsLambda ->
        awsLambda.deleteFunction(request)
        println("$myFunctionName was deleted")
    }
}
```

- Lihat detail API di topik-topik berikut dalam Referensi API AWS SDK For Kotlin.
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Memanggil
  - ListFunctions
  - UpdateFunctionCode
  - UpdateFunctionConfiguration

PHP

SDK untuk PHP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkap dan pelajari cara menyiapkan dan menjalankan di Repositori Contoh Kode AWS.

```
namespace Lambda;

use Aws\S3\S3Client;
use GuzzleHttp\Psr7\Stream;
use Iam\IAMService;

class GettingStartedWithLambda
{
    public function run()
    {
        echo("\n");
        echo("---------------------------------------\n");
        print("Welcome to the AWS Lambda getting started demo using PHP!\n");
        echo("---------------------------------------\n");

        $clientArgs = [
            'region' => 'us-west-2',
            'version' => 'latest',
            'profile' => 'default',
        ];
        $uniqid = uniqid();

        $iamService = new IAMService();
        $s3client = new S3Client($clientArgs);
        $lambdaService = new LambdaService();

        echo "First, let's create a role to run our Lambda code.\n";
        $roleName = "test-lambda-role-$uniqid";
        $rolePolicyDocument = "{
            \"Version\": \"2012-10-17\",
            \"Statement\": [
                {
```

```
                    \"Effect\": \"Allow\",
                    \"Principal\": {
                        \"Service\": \"lambda.amazonaws.com\"
                    },
                    \"Action\": \"sts:AssumeRole\"
                }
            ]
        }";
        $role = $iamService->createRole($roleName, $rolePolicyDocument);
        echo "Created role {$role['RoleName']}.\n";

        $iamService->attachRolePolicy(
            $role['RoleName'],
            "arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole"
        );
        echo "Attached the AWSLambdaBasicExecutionRole to {$role['RoleName']}.
\n";

        echo "\nNow let's create an S3 bucket and upload our Lambda code there.
\n";
        $bucketName = "amzn-s3-demo-bucket-$uniqid";
        $s3client->createBucket([
            'Bucket' => $bucketName,
        ]);
        echo "Created bucket $bucketName.\n";

        $functionName = "doc_example_lambda_$uniqid";
        $codeBasic = __DIR__ . "/lambda_handler_basic.zip";
        $handler = "lambda_handler_basic";
        $file = file_get_contents($codeBasic);
        $s3client->putObject([
            'Bucket' => $bucketName,
            'Key' => $functionName,
            'Body' => $file,
        ]);
        echo "Uploaded the Lambda code.\n";

        $createLambdaFunction = $lambdaService->createFunction($functionName,
 $role, $bucketName, $handler);
        // Wait until the function has finished being created.
        do {
            $getLambdaFunction = $lambdaService-
>getFunction($createLambdaFunction['FunctionName']);
        } while ($getLambdaFunction['Configuration']['State'] == "Pending");
```

```
        echo "Created Lambda function {$getLambdaFunction['Configuration']
['FunctionName']}.\n";

        sleep(1);

        echo "\nOk, let's invoke that Lambda code.\n";
        $basicParams = [
            'action' => 'increment',
            'number' => 3,
        ];
        /** @var Stream $invokeFunction */
        $invokeFunction = $lambdaService->invoke($functionName, $basicParams)
['Payload'];
        $result = json_decode($invokeFunction->getContents())->result;
        echo "After invoking the Lambda code with the input of
 {$basicParams['number']} we received $result.\n";

        echo "\nSince that's working, let's update the Lambda code.\n";
        $codeCalculator = "lambda_handler_calculator.zip";
        $handlerCalculator = "lambda_handler_calculator";
        echo "First, put the new code into the S3 bucket.\n";
        $file = file_get_contents($codeCalculator);
        $s3client->putObject([
            'Bucket' => $bucketName,
            'Key' => $functionName,
            'Body' => $file,
        ]);
        echo "New code uploaded.\n";

        $lambdaService->updateFunctionCode($functionName, $bucketName,
 $functionName);
        // Wait for the Lambda code to finish updating.
        do {
            $getLambdaFunction = $lambdaService-
>getFunction($createLambdaFunction['FunctionName']);
        } while ($getLambdaFunction['Configuration']['LastUpdateStatus'] !==
 "Successful");
        echo "New Lambda code uploaded.\n";

        $environment = [
            'Variable' => ['Variables' => ['LOG_LEVEL' => 'DEBUG']],
        ];
        $lambdaService->updateFunctionConfiguration($functionName,
 $handlerCalculator, $environment);
```

```
        do {
            $getLambdaFunction = $lambdaService-
>getFunction($createLambdaFunction['FunctionName']);
        } while ($getLambdaFunction['Configuration']['LastUpdateStatus'] !==
 "Successful");
        echo "Lambda code updated with new handler and a LOG_LEVEL of DEBUG for
 more information.\n";

        echo "Invoke the new code with some new data.\n";
        $calculatorParams = [
            'action' => 'plus',
            'x' => 5,
            'y' => 4,
        ];
        $invokeFunction = $lambdaService->invoke($functionName,
 $calculatorParams, "Tail");
        $result = json_decode($invokeFunction['Payload']->getContents())->result;
        echo "Indeed, {$calculatorParams['x']} + {$calculatorParams['y']} does
 equal $result.\n";
        echo "Here's the extra debug info: ";
        echo base64_decode($invokeFunction['LogResult']) . "\n";

        echo "\nBut what happens if you try to divide by zero?\n";
        $divZeroParams = [
            'action' => 'divide',
            'x' => 5,
            'y' => 0,
        ];
        $invokeFunction = $lambdaService->invoke($functionName, $divZeroParams,
 "Tail");
        $result = json_decode($invokeFunction['Payload']->getContents())->result;
        echo "You get a |$result| result.\n";
        echo "And an error message: ";
        echo base64_decode($invokeFunction['LogResult']) . "\n";

        echo "\nHere's all the Lambda functions you have in this Region:\n";
        $listLambdaFunctions = $lambdaService->listFunctions(5);
        $allLambdaFunctions = $listLambdaFunctions['Functions'];
        $next = $listLambdaFunctions->get('NextMarker');
        while ($next != false) {
            $listLambdaFunctions = $lambdaService->listFunctions(5, $next);
            $next = $listLambdaFunctions->get('NextMarker');
            $allLambdaFunctions = array_merge($allLambdaFunctions,
 $listLambdaFunctions['Functions']);
```

```
        }
        foreach ($allLambdaFunctions as $function) {
            echo "{$function['FunctionName']}\n";
        }

        echo "\n\nAnd don't forget to clean up your data!\n";

        $lambdaService->deleteFunction($functionName);
        echo "Deleted Lambda function.\n";
        $iamService->deleteRole($role['RoleName']);
        echo "Deleted Role.\n";
        $deleteObjects = $s3client->listObjectsV2([
            'Bucket' => $bucketName,
        ]);
        $deleteObjects = $s3client->deleteObjects([
            'Bucket' => $bucketName,
            'Delete' => [
                'Objects' => $deleteObjects['Contents'],
            ]
        ]);
        echo "Deleted all objects from the S3 bucket.\n";
        $s3client->deleteBucket(['Bucket' => $bucketName]);
        echo "Deleted the bucket.\n";
    }
}
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK untuk PHP .
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Memanggil
  - ListFunctions
  - UpdateFunctionCode
  - UpdateFunctionConfiguration

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Tentukan handler Lambda yang menambah angka.

```
import logging

logger = logging.getLogger()
logger.setLevel(logging.INFO)


def lambda_handler(event, context):
    """
    Accepts an action and a single number, performs the specified action on the
 number,
    and returns the result. The only allowable action is 'increment'.

    :param event: The event dict that contains the parameters sent when the
 function
                  is invoked.
    :param context: The context in which the function is called.
    :return: The result of the action.
    """
    result = None
    action = event.get("action")
    if action == "increment":
        result = event.get("number", 0) + 1
        logger.info("Calculated result of %s", result)
    else:
        logger.error("%s is not a valid action.", action)

    response = {"result": result}
    return response
```

Tentukan handler Lambda kedua yang melakukan operasi aritmatika.

```
import logging
import os


logger = logging.getLogger()

# Define a list of Python lambda functions that are called by this AWS Lambda
 function.
ACTIONS = {
    "plus": lambda x, y: x + y,
    "minus": lambda x, y: x - y,
    "times": lambda x, y: x * y,
    "divided-by": lambda x, y: x / y,
}


def lambda_handler(event, context):
    """
    Accepts an action and two numbers, performs the specified action on the
 numbers,
    and returns the result.

    :param event: The event dict that contains the parameters sent when the
 function
                  is invoked.
    :param context: The context in which the function is called.
    :return: The result of the specified action.
    """
    # Set the log level based on a variable configured in the Lambda environment.
    logger.setLevel(os.environ.get("LOG_LEVEL", logging.INFO))
    logger.debug("Event: %s", event)

    action = event.get("action")
    func = ACTIONS.get(action)
    x = event.get("x")
    y = event.get("y")
    result = None
    try:
        if func is not None and x is not None and y is not None:
```

```
            result = func(x, y)
            logger.info("%s %s %s is %s", x, action, y, result)
        else:
            logger.error("I can't calculate %s %s %s.", x, action, y)
    except ZeroDivisionError:
        logger.warning("I can't divide %s by 0!", x)

    response = {"result": result}
    return response
```

Buat fungsi yang membungkus tindakan Lambda.

```
class LambdaWrapper:
    def __init__(self, lambda_client, iam_resource):
        self.lambda_client = lambda_client
        self.iam_resource = iam_resource


    @staticmethod
    def create_deployment_package(source_file, destination_file):
        """
        Creates a Lambda deployment package in .zip format in an in-memory
 buffer. This
        buffer can be passed directly to Lambda when creating the function.

        :param source_file: The name of the file that contains the Lambda handler
                            function.
        :param destination_file: The name to give the file when it's deployed to
 Lambda.
        :return: The deployment package.
        """
        buffer = io.BytesIO()
        with zipfile.ZipFile(buffer, "w") as zipped:
            zipped.write(source_file, destination_file)
        buffer.seek(0)
        return buffer.read()

    def get_iam_role(self, iam_role_name):
        """
        Get an AWS Identity and Access Management (IAM) role.
```

```
        :param iam_role_name: The name of the role to retrieve.
        :return: The IAM role.
        """
        role = None
        try:
            temp_role = self.iam_resource.Role(iam_role_name)
            temp_role.load()
            role = temp_role
            logger.info("Got IAM role %s", role.name)
        except ClientError as err:
            if err.response["Error"]["Code"] == "NoSuchEntity":
                logger.info("IAM role %s does not exist.", iam_role_name)
            else:
                logger.error(
                    "Couldn't get IAM role %s. Here's why: %s: %s",
                    iam_role_name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        return role

    def create_iam_role_for_lambda(self, iam_role_name):
        """
        Creates an IAM role that grants the Lambda function basic permissions. If
a
        role with the specified name already exists, it is used for the demo.

        :param iam_role_name: The name of the role to create.
        :return: The role and a value that indicates whether the role is newly
created.
        """
        role = self.get_iam_role(iam_role_name)
        if role is not None:
            return role, False

        lambda_assume_role_policy = {
            "Version":"2012-10-17",
            "Statement": [
                {
                    "Effect": "Allow",
                    "Principal": {"Service": "lambda.amazonaws.com"},
                    "Action": "sts:AssumeRole",
```

```
                }
            ],
        }
        policy_arn = "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole"

        try:
            role = self.iam_resource.create_role(
                RoleName=iam_role_name,
                AssumeRolePolicyDocument=json.dumps(lambda_assume_role_policy),
            )
            logger.info("Created role %s.", role.name)
            role.attach_policy(PolicyArn=policy_arn)
            logger.info("Attached basic execution policy to role %s.", role.name)
        except ClientError as error:
            if error.response["Error"]["Code"] == "EntityAlreadyExists":
                role = self.iam_resource.Role(iam_role_name)
                logger.warning("The role %s already exists. Using it.",
 iam_role_name)
            else:
                logger.exception(
                    "Couldn't create role %s or attach policy %s.",
                    iam_role_name,
                    policy_arn,
                )
                raise

        return role, True

    def get_function(self, function_name):
        """
        Gets data about a Lambda function.

        :param function_name: The name of the function.
        :return: The function data.
        """
        response = None
        try:
            response =
 self.lambda_client.get_function(FunctionName=function_name)
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.info("Function %s does not exist.", function_name)
            else:
```

```
                logger.error(
                    "Couldn't get function %s. Here's why: %s: %s",
                    function_name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        return response


    def create_function(
        self, function_name, handler_name, iam_role, deployment_package
    ):
        """
        Deploys a Lambda function.

        :param function_name: The name of the Lambda function.
        :param handler_name: The fully qualified name of the handler function.
This
                             must include the file name and the function name.
        :param iam_role: The IAM role to use for the function.
        :param deployment_package: The deployment package that contains the
function
                                   code in .zip format.
        :return: The Amazon Resource Name (ARN) of the newly created function.
        """
        try:
            response = self.lambda_client.create_function(
                FunctionName=function_name,
                Description="AWS Lambda doc example",
                Runtime="python3.9",
                Role=iam_role.arn,
                Handler=handler_name,
                Code={"ZipFile": deployment_package},
                Publish=True,
            )
            function_arn = response["FunctionArn"]
            waiter = self.lambda_client.get_waiter("function_active_v2")
            waiter.wait(FunctionName=function_name)
            logger.info(
                "Created function '%s' with ARN: '%s'.",
                function_name,
                response["FunctionArn"],
            )
```

```
        except ClientError:
            logger.error("Couldn't create function %s.", function_name)
            raise
        else:
            return function_arn


    def delete_function(self, function_name):
        """
        Deletes a Lambda function.

        :param function_name: The name of the function to delete.
        """
        try:
            self.lambda_client.delete_function(FunctionName=function_name)
        except ClientError:
            logger.exception("Couldn't delete function %s.", function_name)
            raise


    def invoke_function(self, function_name, function_params, get_log=False):
        """
        Invokes a Lambda function.

        :param function_name: The name of the function to invoke.
        :param function_params: The parameters of the function as a dict. This
dict
                                is serialized to JSON before it is sent to
Lambda.
        :param get_log: When true, the last 4 KB of the execution log are
included in
                        the response.
        :return: The response from the function invocation.
        """
        try:
            response = self.lambda_client.invoke(
                FunctionName=function_name,
                Payload=json.dumps(function_params),
                LogType="Tail" if get_log else "None",
            )
            logger.info("Invoked function %s.", function_name)
        except ClientError:
            logger.exception("Couldn't invoke function %s.", function_name)
            raise
```

```
        return response


    def update_function_code(self, function_name, deployment_package):
        """
        Updates the code for a Lambda function by submitting a .zip archive that
contains
        the code for the function.

        :param function_name: The name of the function to update.
        :param deployment_package: The function code to update, packaged as bytes
in
                                   .zip format.
        :return: Data about the update, including the status.
        """
        try:
            response = self.lambda_client.update_function_code(
                FunctionName=function_name, ZipFile=deployment_package
            )
        except ClientError as err:
            logger.error(
                "Couldn't update function %s. Here's why: %s: %s",
                function_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response


    def update_function_configuration(self, function_name, env_vars):
        """
        Updates the environment variables for a Lambda function.

        :param function_name: The name of the function to update.
        :param env_vars: A dict of environment variables to update.
        :return: Data about the update, including the status.
        """
        try:
            response = self.lambda_client.update_function_configuration(
                FunctionName=function_name, Environment={"Variables": env_vars}
            )
        except ClientError as err:
```

```
            logger.error(
                "Couldn't update function configuration %s. Here's why: %s: %s",
                function_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response


    def list_functions(self):
        """
        Lists the Lambda functions for the current account.
        """
        try:
            func_paginator = self.lambda_client.get_paginator("list_functions")
            for func_page in func_paginator.paginate():
                for func in func_page["Functions"]:
                    print(func["FunctionName"])
                    desc = func.get("Description")
                    if desc:
                        print(f"\t{desc}")
                    print(f"\t{func['Runtime']}: {func['Handler']}")
        except ClientError as err:
            logger.error(
                "Couldn't list functions. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

Buat fungsi yang menjalankan skenario.

```
class UpdateFunctionWaiter(CustomWaiter):
    """A custom waiter that waits until a function is successfully updated."""

    def __init__(self, client):
        super().__init__(
```

```
            "UpdateSuccess",
            "GetFunction",
            "Configuration.LastUpdateStatus",
            {"Successful": WaitState.SUCCESS, "Failed": WaitState.FAILURE},
            client,
        )

    def wait(self, function_name):
        self._wait(FunctionName=function_name)


def run_scenario(lambda_client, iam_resource, basic_file, calculator_file,
 lambda_name):
    """
    Runs the scenario.

    :param lambda_client: A Boto3 Lambda client.
    :param iam_resource: A Boto3 IAM resource.
    :param basic_file: The name of the file that contains the basic Lambda
 handler.
    :param calculator_file: The name of the file that contains the calculator
 Lambda handler.
    :param lambda_name: The name to give resources created for the scenario, such
 as the
                        IAM role and the Lambda function.
    """
    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")

    print("-" * 88)
    print("Welcome to the AWS Lambda getting started with functions demo.")
    print("-" * 88)

    wrapper = LambdaWrapper(lambda_client, iam_resource)

    print("Checking for IAM role for Lambda...")
    iam_role, should_wait = wrapper.create_iam_role_for_lambda(lambda_name)
    if should_wait:
        logger.info("Giving AWS time to create resources...")
        wait(10)

    print(f"Looking for function {lambda_name}...")
    function = wrapper.get_function(lambda_name)
    if function is None:
        print("Zipping the Python script into a deployment package...")
```

```
        deployment_package = wrapper.create_deployment_package(
            basic_file, f"{lambda_name}.py"
        )
        print(f"...and creating the {lambda_name} Lambda function.")
        wrapper.create_function(
            lambda_name, f"{lambda_name}.lambda_handler", iam_role,
deployment_package
        )
    else:
        print(f"Function {lambda_name} already exists.")
    print("-" * 88)

    print(f"Let's invoke {lambda_name}. This function increments a number.")
    action_params = {
        "action": "increment",
        "number": q.ask("Give me a number to increment: ", q.is_int),
    }
    print(f"Invoking {lambda_name}...")
    response = wrapper.invoke_function(lambda_name, action_params)
    print(
        f"Incrementing {action_params['number']} resulted in "
        f"{json.load(response['Payload'])}"
    )
    print("-" * 88)

    print(f"Let's update the function to an arithmetic calculator.")
    q.ask("Press Enter when you're ready.")
    print("Creating a new deployment package...")
    deployment_package = wrapper.create_deployment_package(
        calculator_file, f"{lambda_name}.py"
    )
    print(f"...and updating the {lambda_name} Lambda function.")
    update_waiter = UpdateFunctionWaiter(lambda_client)
    wrapper.update_function_code(lambda_name, deployment_package)
    update_waiter.wait(lambda_name)
    print(f"This function uses an environment variable to control logging
level.")
    print(f"Let's set it to DEBUG to get the most logging.")
    wrapper.update_function_configuration(
        lambda_name, {"LOG_LEVEL": logging.getLevelName(logging.DEBUG)}
    )

    actions = ["plus", "minus", "times", "divided-by"]
    want_invoke = True
```

```
    while want_invoke:
        print(f"Let's invoke {lambda_name}. You can invoke these actions:")
        for index, action in enumerate(actions):
            print(f"{index + 1}: {action}")
        action_params = {}
        action_index = q.ask(
            "Enter the number of the action you want to take: ",
            q.is_int,
            q.in_range(1, len(actions)),
        )
        action_params["action"] = actions[action_index - 1]
        print(f"You've chosen to invoke 'x {action_params['action']} y'.")
        action_params["x"] = q.ask("Enter a value for x: ", q.is_int)
        action_params["y"] = q.ask("Enter a value for y: ", q.is_int)
        print(f"Invoking {lambda_name}...")
        response = wrapper.invoke_function(lambda_name, action_params, True)
        print(
            f"Calculating {action_params['x']} {action_params['action']}
{action_params['y']} "
            f"resulted in {json.load(response['Payload'])}"
        )
        q.ask("Press Enter to see the logs from the call.")
        print(base64.b64decode(response["LogResult"]).decode())
        want_invoke = q.ask("That was fun. Shall we do it again? (y/n) ",
q.is_yesno)
    print("-" * 88)

    if q.ask(
        "Do you want to list all of the functions in your account? (y/n) ",
q.is_yesno
    ):
        wrapper.list_functions()
    print("-" * 88)

    if q.ask("Ready to delete the function and role? (y/n) ", q.is_yesno):
        for policy in iam_role.attached_policies.all():
            policy.detach_role(RoleName=iam_role.name)
        iam_role.delete()
        print(f"Deleted role {lambda_name}.")
        wrapper.delete_function(lambda_name)
        print(f"Deleted function {lambda_name}.")

    print("\nThanks for watching!")
    print("-" * 88)
```

```
if __name__ == "__main__":
    try:
        run_scenario(
            boto3.client("lambda"),
            boto3.resource("iam"),
            "lambda_handler_basic.py",
            "lambda_handler_calculator.py",
            "doc_example_lambda_calculator",
        )
    except Exception:
        logging.exception("Something went wrong with the demo!")
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK untuk Python (Boto3).
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Memanggil
  - ListFunctions
  - UpdateFunctionCode
  - UpdateFunctionConfiguration

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Siapkan izin IAM prasyarat untuk fungsi Lambda yang mampu menulis log.

```
  # Get an AWS Identity and Access Management (IAM) role.
  #
  # @param iam_role_name: The name of the role to retrieve.
```

```
  # @param action: Whether to create or destroy the IAM apparatus.
  # @return: The IAM role.
  def manage_iam(iam_role_name, action)
    case action
    when 'create'
      create_iam_role(iam_role_name)
    when 'destroy'
      destroy_iam_role(iam_role_name)
    else
      raise "Incorrect action provided. Must provide 'create' or 'destroy'"
    end
  end

  private

  def create_iam_role(iam_role_name)
    role_policy = {
      'Version': '2012-10-17',
      'Statement': [
        {
          'Effect': 'Allow',
          'Principal': { 'Service': 'lambda.amazonaws.com' },
          'Action': 'sts:AssumeRole'
        }
      ]
    }
    role = @iam_client.create_role(
      role_name: iam_role_name,
      assume_role_policy_document: role_policy.to_json
    )
    @iam_client.attach_role_policy(
      {
        policy_arn: 'arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole',
        role_name: iam_role_name
      }
    )
    wait_for_role_to_exist(iam_role_name)
    @logger.debug("Successfully created IAM role: #{role['role']['arn']}")
    sleep(10)
    [role, role_policy.to_json]
  end

  def destroy_iam_role(iam_role_name)
```

```
    @iam_client.detach_role_policy(
      {
        policy_arn: 'arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole',
        role_name: iam_role_name
      }
    )
    @iam_client.delete_role(role_name: iam_role_name)
    @logger.debug("Detached policy & deleted IAM role: #{iam_role_name}")
  end

  def wait_for_role_to_exist(iam_role_name)
    @iam_client.wait_until(:role_exists, { role_name: iam_role_name }) do |w|
      w.max_attempts = 5
      w.delay = 5
    end
  end
```

Tentukan handler Lambda yang menambah angka yang disediakan sebagai parameter pemanggilan.

```
require 'logger'

# A function that increments a whole number by one (1) and logs the result.
# Requires a manually-provided runtime parameter, 'number', which must be Int
#
# @param event [Hash] Parameters sent when the function is invoked
# @param context [Hash] Methods and properties that provide information
# about the invocation, function, and execution environment.
# @return incremented_number [String] The incremented number.
def lambda_handler(event:, context:)
  logger = Logger.new($stdout)
  log_level = ENV['LOG_LEVEL']
  logger.level = case log_level
                 when 'debug'
                   Logger::DEBUG
                 when 'info'
                   Logger::INFO
                 else
                   Logger::ERROR
                 end
  logger.debug('This is a debug log message.')
```

```
  logger.info('This is an info log message. Code executed successfully!')
  number = event['number'].to_i
  incremented_number = number + 1
  logger.info("You provided #{number.round} and it was incremented to
 #{incremented_number.round}")
  incremented_number.round.to_s
end
```

Masukkan fungsi Lambda Anda ke dalam paket penerapan.

```
  # Creates a Lambda deployment package in .zip format.
  #
  # @param source_file: The name of the object, without suffix, for the Lambda
 file and zip.
  # @return: The deployment package.
  def create_deployment_package(source_file)
    Dir.chdir(File.dirname(__FILE__))
    if File.exist?('lambda_function.zip')
      File.delete('lambda_function.zip')
      @logger.debug('Deleting old zip: lambda_function.zip')
    end
    Zip::File.open('lambda_function.zip', create: true) do |zipfile|
      zipfile.add('lambda_function.rb', "#{source_file}.rb")
    end
    @logger.debug("Zipping #{source_file}.rb into: lambda_function.zip.")
    File.read('lambda_function.zip').to_s
  rescue StandardError => e
    @logger.error("There was an error creating deployment package:\n
 #{e.message}")
  end
```

Buat fungsi Lambda baru.

```
  # Deploys a Lambda function.
  #
  # @param function_name: The name of the Lambda function.
  # @param handler_name: The fully qualified name of the handler function.
  # @param role_arn: The IAM role to use for the function.
  # @param deployment_package: The deployment package that contains the function
 code in .zip format.
  # @return: The Amazon Resource Name (ARN) of the newly created function.
```

```
 def create_function(function_name, handler_name, role_arn, deployment_package)
   response = @lambda_client.create_function({
                                               role: role_arn.to_s,
                                               function_name: function_name,
                                               handler: handler_name,
                                               runtime: 'ruby2.7',
                                               code: {
                                                 zip_file: deployment_package
                                               },
                                               environment: {
                                                 variables: {
                                                   'LOG_LEVEL' => 'info'
                                                 }
                                               }
                                             })
   @lambda_client.wait_until(:function_active_v2, { function_name:
function_name }) do |w|
     w.max_attempts = 5
     w.delay = 5
   end
   response
 rescue Aws::Lambda::Errors::ServiceException => e
   @logger.error("There was an error creating #{function_name}:\n #{e.message}")
 rescue Aws::Waiters::Errors::WaiterFailed => e
   @logger.error("Failed waiting for #{function_name} to activate:\n
#{e.message}")
 end
```

Memanggil fungsi Lambda Anda dengan parameter runtime opsional.

```
 # Invokes a Lambda function.
 # @param function_name [String] The name of the function to invoke.
 # @param payload [nil] Payload containing runtime parameters.
 # @return [Object] The response from the function invocation.
 def invoke_function(function_name, payload = nil)
   params = { function_name: function_name }
   params[:payload] = payload unless payload.nil?
   @lambda_client.invoke(params)
 rescue Aws::Lambda::Errors::ServiceException => e
   @logger.error("There was an error executing #{function_name}:\n
#{e.message}")
 end
```

Perbarui konfigurasi fungsi Lambda Anda untuk menyuntikkan variabel lingkungan baru.

```
 # Updates the environment variables for a Lambda function.
 # @param function_name: The name of the function to update.
 # @param log_level: The log level of the function.
 # @return: Data about the update, including the status.
 def update_function_configuration(function_name, log_level)
   @lambda_client.update_function_configuration({
                                                  function_name: function_name,
                                                  environment: {
                                                    variables: {
                                                      'LOG_LEVEL' => log_level
                                                    }
                                                  }
                                                })
   @lambda_client.wait_until(:function_updated_v2, { function_name:
function_name }) do |w|
     w.max_attempts = 5
     w.delay = 5
   end
 rescue Aws::Lambda::Errors::ServiceException => e
   @logger.error("There was an error updating configurations for
#{function_name}:\n #{e.message}")
 rescue Aws::Waiters::Errors::WaiterFailed => e
   @logger.error("Failed waiting for #{function_name} to activate:\n
#{e.message}")
 end
```

Perbarui kode fungsi Lambda Anda dengan paket penerapan berbeda yang berisi kode berbeda.

```
 # Updates the code for a Lambda function by submitting a .zip archive that
contains
 # the code for the function.
 #
 # @param function_name: The name of the function to update.
 # @param deployment_package: The function code to update, packaged as bytes in
 #                            .zip format.
 # @return: Data about the update, including the status.
 def update_function_code(function_name, deployment_package)
```

```
    @lambda_client.update_function_code(
      function_name: function_name,
      zip_file: deployment_package
    )
    @lambda_client.wait_until(:function_updated_v2, { function_name:
function_name }) do |w|
      w.max_attempts = 5
      w.delay = 5
    end
  rescue Aws::Lambda::Errors::ServiceException => e
    @logger.error("There was an error updating function code for:
#{function_name}:\n #{e.message}")
    nil
  rescue Aws::Waiters::Errors::WaiterFailed => e
    @logger.error("Failed waiting for #{function_name} to update:\n
#{e.message}")
  end
```

Daftar semua fungsi Lambda yang ada menggunakan paginator bawaan.

```
  # Lists the Lambda functions for the current account.
  def list_functions
    functions = []
    @lambda_client.list_functions.each do |response|
      response['functions'].each do |function|
        functions.append(function['function_name'])
      end
    end
    functions
  rescue Aws::Lambda::Errors::ServiceException => e
    @logger.error("There was an error listing functions:\n #{e.message}")
  end
```

Hapus fungsi Lambda tertentu.

```
  # Deletes a Lambda function.
  # @param function_name: The name of the function to delete.
  def delete_function(function_name)
    print "Deleting function: #{function_name}..."
    @lambda_client.delete_function(
      function_name: function_name
```

```
    )
    print 'Done!'.green
  rescue Aws::Lambda::Errors::ServiceException => e
    @logger.error("There was an error deleting #{function_name}:\n #{e.message}")
  end
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK untuk Ruby .
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Memanggil
  - ListFunctions
  - UpdateFunctionCode
  - UpdateFunctionConfiguration

Rust

SDK for Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

The Cargo.toml dengan dependensi yang digunakan dalam skenario ini.

```
[package]
name = "lambda-code-examples"
version = "0.1.0"
edition = "2021"

# See more keys and their definitions at https://doc.rust-lang.org/cargo/
reference/manifest.html

[dependencies]
aws-config = { version = "1.0.1", features = ["behavior-version-latest"] }
aws-sdk-ec2 = { version = "1.3.0" }
```

```
aws-sdk-iam = { version = "1.3.0" }
aws-sdk-lambda = { version = "1.3.0" }
aws-sdk-s3 = { version = "1.4.0" }
aws-smithy-types = { version = "1.0.1" }
aws-types = { version = "1.0.1" }
clap = { version = "4.4", features = ["derive"] }
tokio = { version = "1.20.1", features = ["full"] }
tracing-subscriber = { version = "0.3.15", features = ["env-filter"] }
tracing = "0.1.37"
serde_json = "1.0.94"
anyhow = "1.0.71"
uuid = { version = "1.3.3", features = ["v4"] }
lambda_runtime = "0.8.0"
serde = "1.0.164"
```

Kumpulan utilitas yang merampingkan panggilan Lambda untuk skenario ini. File ini adalah src/ations.rs di peti.

```
use anyhow::anyhow;
use aws_sdk_iam::operation::{create_role::CreateRoleError,
 delete_role::DeleteRoleOutput};
use aws_sdk_lambda::{
    operation::{
        delete_function::DeleteFunctionOutput, get_function::GetFunctionOutput,
        invoke::InvokeOutput, list_functions::ListFunctionsOutput,
        update_function_code::UpdateFunctionCodeOutput,
        update_function_configuration::UpdateFunctionConfigurationOutput,
    },
    primitives::ByteStream,
    types::{Environment, FunctionCode, LastUpdateStatus, State},
};
use aws_sdk_s3::{
    error::ErrorMetadata,
    operation::{delete_bucket::DeleteBucketOutput,
 delete_object::DeleteObjectOutput},
    types::CreateBucketConfiguration,
};
use aws_smithy_types::Blob;
use serde::{ser::SerializeMap, Serialize};
use std::{fmt::Display, path::PathBuf, str::FromStr, time::Duration};
use tracing::{debug, info, warn};
```

```
/* Operation describes  */
#[derive(Clone, Copy, Debug, Serialize)]
pub enum Operation {
    #[serde(rename = "plus")]
    Plus,
    #[serde(rename = "minus")]
    Minus,
    #[serde(rename = "times")]
    Times,
    #[serde(rename = "divided-by")]
    DividedBy,
}

impl FromStr for Operation {
    type Err = anyhow::Error;

    fn from_str(s: &str) -> Result<Self, Self::Err> {
        match s {
            "plus" => Ok(Operation::Plus),
            "minus" => Ok(Operation::Minus),
            "times" => Ok(Operation::Times),
            "divided-by" => Ok(Operation::DividedBy),
            _ => Err(anyhow!("Unknown operation {s}")),
        }
    }
}

impl Display for Operation {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        match self {
            Operation::Plus => write!(f, "plus"),
            Operation::Minus => write!(f, "minus"),
            Operation::Times => write!(f, "times"),
            Operation::DividedBy => write!(f, "divided-by"),
        }
    }
}

/**
 * InvokeArgs will be serialized as JSON and sent to the AWS Lambda handler.
 */
#[derive(Debug)]
pub enum InvokeArgs {
```

```
    Increment(i32),
    Arithmetic(Operation, i32, i32),
}

impl Serialize for InvokeArgs {
    fn serialize<S>(&self, serializer: S) -> Result<S::Ok, S::Error>
    where
        S: serde::Serializer,
    {
        match self {
            InvokeArgs::Increment(i) => serializer.serialize_i32(*i),
            InvokeArgs::Arithmetic(o, i, j) => {
                let mut map: S::SerializeMap =
 serializer.serialize_map(Some(3))?;
                map.serialize_key(&"op".to_string())?;
                map.serialize_value(&o.to_string())?;
                map.serialize_key(&"i".to_string())?;
                map.serialize_value(&i)?;
                map.serialize_key(&"j".to_string())?;
                map.serialize_value(&j)?;
                map.end()
            }
        }
    }
}

/** A policy document allowing Lambda to execute this function on the account's
 behalf. */
const ROLE_POLICY_DOCUMENT: &str = r#"{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": { "Service": "lambda.amazonaws.com" },
            "Action": "sts:AssumeRole"
        }
    ]
}"#;

/**
 * A LambdaManager gathers all the resources necessary to run the Lambda example
 scenario.
 * This includes instantiated aws_sdk clients and details of resource names.
 */
```

```
pub struct LambdaManager {
    iam_client: aws_sdk_iam::Client,
    lambda_client: aws_sdk_lambda::Client,
    s3_client: aws_sdk_s3::Client,
    lambda_name: String,
    role_name: String,
    bucket: String,
    own_bucket: bool,
}

// These unit type structs provide nominal typing on top of String parameters for
 LambdaManager::new
pub struct LambdaName(pub String);
pub struct RoleName(pub String);
pub struct Bucket(pub String);
pub struct OwnBucket(pub bool);

impl LambdaManager {
    pub fn new(
        iam_client: aws_sdk_iam::Client,
        lambda_client: aws_sdk_lambda::Client,
        s3_client: aws_sdk_s3::Client,
        lambda_name: LambdaName,
        role_name: RoleName,
        bucket: Bucket,
        own_bucket: OwnBucket,
    ) -> Self {
        Self {
            iam_client,
            lambda_client,
            s3_client,
            lambda_name: lambda_name.0,
            role_name: role_name.0,
            bucket: bucket.0,
            own_bucket: own_bucket.0,
        }
    }

    /**
     * Load the AWS configuration from the environment.
     * Look up lambda_name and bucket if none are given, or generate a random
 name if not present in the environment.
     * If the bucket name is provided, the caller needs to have created the
 bucket.
```

```
    * If the bucket name is generated, it will be created.
    */
   pub async fn load_from_env(lambda_name: Option<String>, bucket:
Option<String>) -> Self {
       let sdk_config = aws_config::load_from_env().await;
       let lambda_name = LambdaName(lambda_name.unwrap_or_else(|| {
           std::env::var("LAMBDA_NAME").unwrap_or_else(|_|
"rust_lambda_example".to_string())
       }));
       let role_name = RoleName(format!("{}_role", lambda_name.0));
       let (bucket, own_bucket) =
           match bucket {
               Some(bucket) => (Bucket(bucket), false),
               None => (
                   Bucket(std::env::var("LAMBDA_BUCKET").unwrap_or_else(|_| {
                       format!("rust-lambda-example-{}", uuid::Uuid::new_v4())
                   })),
                   true,
               ),
           };

       let s3_client = aws_sdk_s3::Client::new(&sdk_config);

       if own_bucket {
           info!("Creating bucket for demo: {}", bucket.0);
           s3_client
               .create_bucket()
               .bucket(bucket.0.clone())
               .create_bucket_configuration(
                   CreateBucketConfiguration::builder()

.location_constraint(aws_sdk_s3::types::BucketLocationConstraint::from(
                           sdk_config.region().unwrap().as_ref(),
                       ))
                       .build(),
               )
               .send()
               .await
               .unwrap();
       }

       Self::new(
           aws_sdk_iam::Client::new(&sdk_config),
           aws_sdk_lambda::Client::new(&sdk_config),
```

```
            s3_client,
            lambda_name,
            role_name,
            bucket,
            OwnBucket(own_bucket),
        )
    }

    /**
     * Upload function code from a path to a zip file.
     * The zip file must have an AL2 Linux-compatible binary called `bootstrap`.
     * The easiest way to create such a zip is to use `cargo lambda build --
output-format Zip`.
     */
    async fn prepare_function(
        &self,
        zip_file: PathBuf,
        key: Option<String>,
    ) -> Result<FunctionCode, anyhow::Error> {
        let body = ByteStream::from_path(zip_file).await?;

        let key = key.unwrap_or_else(|| format!("{}_code", self.lambda_name));

        info!("Uploading function code to s3://{}/{}", self.bucket, key);
        let _ = self
            .s3_client
            .put_object()
            .bucket(self.bucket.clone())
            .key(key.clone())
            .body(body)
            .send()
            .await?;

        Ok(FunctionCode::builder()
            .s3_bucket(self.bucket.clone())
            .s3_key(key)
            .build())
    }

    /**
     * Create a function, uploading from a zip file.
     */
    pub async fn create_function(&self, zip_file: PathBuf) -> Result<String,
 anyhow::Error> {
```

```
        let code = self.prepare_function(zip_file, None).await?;

        let key = code.s3_key().unwrap().to_string();

        let role = self.create_role().await.map_err(|e| anyhow!(e))?;

        info!("Created iam role, waiting 15s for it to become active");
        tokio::time::sleep(Duration::from_secs(15)).await;

        info!("Creating lambda function {}", self.lambda_name);
        let _ = self
            .lambda_client
            .create_function()
            .function_name(self.lambda_name.clone())
            .code(code)
            .role(role.arn())
            .runtime(aws_sdk_lambda::types::Runtime::Providedal2)
            .handler("_unused")
            .send()
            .await
            .map_err(anyhow::Error::from)?;

        self.wait_for_function_ready().await?;

        self.lambda_client
            .publish_version()
            .function_name(self.lambda_name.clone())
            .send()
            .await?;

        Ok(key)
    }

    /**
     * Create an IAM execution role for the managed Lambda function.
     * If the role already exists, use that instead.
     */
    async fn create_role(&self) -> Result<aws_sdk_iam::types::Role,
CreateRoleError> {
        info!("Creating execution role for function");
        let get_role = self
            .iam_client
            .get_role()
            .role_name(self.role_name.clone())
```

```
            .send()
            .await;
        if let Ok(get_role) = get_role {
            if let Some(role) = get_role.role {
                return Ok(role);
            }
        }

        let create_role = self
            .iam_client
            .create_role()
            .role_name(self.role_name.clone())
            .assume_role_policy_document(ROLE_POLICY_DOCUMENT)
            .send()
            .await;

        match create_role {
            Ok(create_role) => match create_role.role {
                Some(role) => Ok(role),
                None => Err(CreateRoleError::generic(
                    ErrorMetadata::builder()
                        .message("CreateRole returned empty success")
                        .build(),
                )),
            },
            Err(err) => Err(err.into_service_error()),
        }
    }

    /**
     * Poll `is_function_ready` with a 1-second delay. It returns when the
function is ready or when there's an error checking the function's state.
     */
    pub async fn wait_for_function_ready(&self) -> Result<(), anyhow::Error> {
        info!("Waiting for function");
        while !self.is_function_ready(None).await? {
            info!("Function is not ready, sleeping 1s");
            tokio::time::sleep(Duration::from_secs(1)).await;
        }
        Ok(())
    }

    /**
     * Check if a Lambda function is ready to be invoked.
```

```
    * A Lambda function is ready for this scenario when its state is active and
its LastUpdateStatus is Successful.
    * Additionally, if a sha256 is provided, the function must have that as its
current code hash.
    * Any missing properties or failed requests will be reported as an Err.
    */
   async fn is_function_ready(
       &self,
       expected_code_sha256: Option<&str>,
   ) -> Result<bool, anyhow::Error> {
       match self.get_function().await {
           Ok(func) => {
               if let Some(config) = func.configuration() {
                   if let Some(state) = config.state() {
                       info!(?state, "Checking if function is active");
                       if !matches!(state, State::Active) {
                           return Ok(false);
                       }
                   }
                   match config.last_update_status() {
                       Some(last_update_status) => {
                           info!(?last_update_status, "Checking if function is
ready");
                           match last_update_status {
                               LastUpdateStatus::Successful => {
                                   // continue
                               }
                               LastUpdateStatus::Failed |
LastUpdateStatus::InProgress => {
                                   return Ok(false);
                               }
                               unknown => {
                                   warn!(
                                       status_variant = unknown.as_str(),
                                       "LastUpdateStatus unknown"
                                   );
                                   return Err(anyhow!(
                                       "Unknown LastUpdateStatus, fn config is
{config:?}"
                                   ));
                               }
                           }
                       }
                       None => {
```

```
                            warn!("Missing last update status");
                            return Ok(false);
                        }
                    };
                    if expected_code_sha256.is_none() {
                        return Ok(true);
                    }
                    if let Some(code_sha256) = config.code_sha256() {
                        return Ok(code_sha256 ==
expected_code_sha256.unwrap_or_default());
                    }
                }
            }
            Err(e) => {
                warn!(?e, "Could not get function while waiting");
            }
        }
        Ok(false)
    }

    /** Get the Lambda function with this Manager's name. */
    pub async fn get_function(&self) -> Result<GetFunctionOutput, anyhow::Error>
{
        info!("Getting lambda function");
        self.lambda_client
            .get_function()
            .function_name(self.lambda_name.clone())
            .send()
            .await
            .map_err(anyhow::Error::from)
    }

    /** List all Lambda functions in the current Region. */
    pub async fn list_functions(&self) -> Result<ListFunctionsOutput,
anyhow::Error> {
        info!("Listing lambda functions");
        self.lambda_client
            .list_functions()
            .send()
            .await
            .map_err(anyhow::Error::from)
    }

    /** Invoke the lambda function using calculator InvokeArgs. */
```

```
    pub async fn invoke(&self, args: InvokeArgs) -> Result<InvokeOutput,
anyhow::Error> {
        info!(?args, "Invoking {}", self.lambda_name);
        let payload = serde_json::to_string(&args)?;
        debug!(?payload, "Sending payload");
        self.lambda_client
            .invoke()
            .function_name(self.lambda_name.clone())
            .payload(Blob::new(payload))
            .send()
            .await
            .map_err(anyhow::Error::from)
    }

    /** Given a Path to a zip file, update the function's code and wait for the
update to finish. */
    pub async fn update_function_code(
        &self,
        zip_file: PathBuf,
        key: String,
    ) -> Result<UpdateFunctionCodeOutput, anyhow::Error> {
        let function_code = self.prepare_function(zip_file, Some(key)).await?;

        info!("Updating code for {}", self.lambda_name);
        let update = self
            .lambda_client
            .update_function_code()
            .function_name(self.lambda_name.clone())
            .s3_bucket(self.bucket.clone())
            .s3_key(function_code.s3_key().unwrap().to_string())
            .send()
            .await
            .map_err(anyhow::Error::from)?;

        self.wait_for_function_ready().await?;

        Ok(update)
    }

    /** Update the environment for a function. */
    pub async fn update_function_configuration(
        &self,
        environment: Environment,
    ) -> Result<UpdateFunctionConfigurationOutput, anyhow::Error> {
```

```
        info!(
            ?environment,
            "Updating environment for {}", self.lambda_name
        );
        let updated = self
            .lambda_client
            .update_function_configuration()
            .function_name(self.lambda_name.clone())
            .environment(environment)
            .send()
            .await
            .map_err(anyhow::Error::from)?;

        self.wait_for_function_ready().await?;

        Ok(updated)
    }

    /** Delete a function and its role, and if possible or necessary, its
associated code object and bucket. */
    pub async fn delete_function(
        &self,
        location: Option<String>,
    ) -> (
        Result<DeleteFunctionOutput, anyhow::Error>,
        Result<DeleteRoleOutput, anyhow::Error>,
        Option<Result<DeleteObjectOutput, anyhow::Error>>,
    ) {
        info!("Deleting lambda function {}", self.lambda_name);
        let delete_function = self
            .lambda_client
            .delete_function()
            .function_name(self.lambda_name.clone())
            .send()
            .await
            .map_err(anyhow::Error::from);

        info!("Deleting iam role {}", self.role_name);
        let delete_role = self
            .iam_client
            .delete_role()
            .role_name(self.role_name.clone())
            .send()
            .await
```

```
            .map_err(anyhow::Error::from);

        let delete_object: Option<Result<DeleteObjectOutput, anyhow::Error>> =
            if let Some(location) = location {
                info!("Deleting object {location}");
                Some(
                    self.s3_client
                        .delete_object()
                        .bucket(self.bucket.clone())
                        .key(location)
                        .send()
                        .await
                        .map_err(anyhow::Error::from),
                )
            } else {
                info!(?location, "Skipping delete object");
                None
            };

        (delete_function, delete_role, delete_object)
    }

    pub async fn cleanup(
        &self,
        location: Option<String>,
    ) -> (
        (
            Result<DeleteFunctionOutput, anyhow::Error>,
            Result<DeleteRoleOutput, anyhow::Error>,
            Option<Result<DeleteObjectOutput, anyhow::Error>>,
        ),
        Option<Result<DeleteBucketOutput, anyhow::Error>>,
    ) {
        let delete_function = self.delete_function(location).await;

        let delete_bucket = if self.own_bucket {
            info!("Deleting bucket {}", self.bucket);
            if delete_function.2.is_none() ||
delete_function.2.as_ref().unwrap().is_ok() {
                Some(
                    self.s3_client
                        .delete_bucket()
                        .bucket(self.bucket.clone())
                        .send()
```

```
                        .await
                        .map_err(anyhow::Error::from),
                )
            } else {
                None
            }
        } else {
            info!("No bucket to clean up");
            None
        };

        (delete_function, delete_bucket)
    }
}

/**
 * Testing occurs primarily as an integration test running the `scenario` bin
 successfully.
 * Each action relies deeply on the internal workings and state of Amazon Simple
 Storage Service (Amazon S3), Lambda, and IAM working together.
 * It is therefore infeasible to mock the clients to test the individual actions.
 */
#[cfg(test)]
mod test {
    use super::{InvokeArgs, Operation};
    use serde_json::json;

    /** Make sure that the JSON output of serializing InvokeArgs is what's
 expected by the calculator. */
    #[test]
    fn test_serialize() {
        assert_eq!(json!(InvokeArgs::Increment(5)), 5);
        assert_eq!(
            json!(InvokeArgs::Arithmetic(Operation::Plus, 5, 7)).to_string(),
            r#"{"op":"plus","i":5,"j":7}"#.to_string(),
        );
    }
}
```

Sebuah biner untuk menjalankan skenario dari depan ke ujung, menggunakan flag baris perintah untuk mengontrol beberapa perilaku. File ini adalah src/bin/scenario.rs di peti.

```
/*
## Service actions

Service actions wrap the SDK call, taking a client and any specific parameters
 necessary for the call.

* CreateFunction
* GetFunction
* ListFunctions
* Invoke
* UpdateFunctionCode
* UpdateFunctionConfiguration
* DeleteFunction

## Scenario
A scenario runs at a command prompt and prints output to the user on the result
 of each service action. A scenario can run in one of two ways: straight through,
 printing out progress as it goes, or as an interactive question/answer script.

## Getting started with functions

Use an SDK to manage AWS Lambda functions: create a function, invoke it, update
 its code, invoke it again, view its output and logs, and delete it.

This scenario uses two Lambda handlers:
_Note: Handlers don't use AWS SDK API calls._

The increment handler is straightforward:

1. It accepts a number, increments it, and returns the new value.
2. It performs simple logging of the result.

The arithmetic handler is more complex:
1. It accepts a set of actions ['plus', 'minus', 'times', 'divided-by'] and two
 numbers, and returns the result of the calculation.
2. It uses an environment variable to control log level (such as DEBUG, INFO,
 WARNING, ERROR).
It logs a few things at different levels, such as:
    * DEBUG: Full event data.
    * INFO: The calculation result.
    * WARN~ING~: When a divide by zero error occurs.
    * This will be the typical `RUST_LOG` variable.
```

```
The steps of the scenario are:

1. Create an AWS Identity and Access Management (IAM) role that meets the
 following requirements:
    * Has an assume_role policy that grants 'lambda.amazonaws.com' the
 'sts:AssumeRole' action.
    * Attaches the 'arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole' managed role.
    * _You must wait for ~10 seconds after the role is created before you can use
 it!_
2. Create a function (CreateFunction) for the increment handler by packaging it
 as a zip and doing one of the following:
    * Adding it with CreateFunction Code.ZipFile.
    * --or--
    * Uploading it to Amazon Simple Storage Service (Amazon S3) and adding it
 with CreateFunction Code.S3Bucket/S3Key.
    * _Note: Zipping the file does not have to be done in code._
    * If you have a waiter, use it to wait until the function is active.
 Otherwise, call GetFunction until State is Active.
3. Invoke the function with a number and print the result.
4. Update the function (UpdateFunctionCode) to the arithmetic handler by
 packaging it as a zip and doing one of the following:
    * Adding it with UpdateFunctionCode ZipFile.
    * --or--
    * Uploading it to Amazon S3 and adding it with UpdateFunctionCode S3Bucket/
S3Key.
5. Call GetFunction until Configuration.LastUpdateStatus is 'Successful' (or
 'Failed').
6. Update the environment variable by calling UpdateFunctionConfiguration and
 pass it a log level, such as:
    * Environment={'Variables': {'RUST_LOG': 'TRACE'}}
7. Invoke the function with an action from the list and a couple of values.
 Include LogType='Tail' to get logs in the result. Print the result of the
 calculation and the log.
8. [Optional] Invoke the function to provoke a divide-by-zero error and show the
 log result.
9. List all functions for the account, using pagination (ListFunctions).
10. Delete the function (DeleteFunction).
11. Delete the role.

Each step should use the function created in Service Actions to abstract calling
 the SDK.
```

```
 */

use aws_sdk_lambda::{operation::invoke::InvokeOutput, types::Environment};
use clap::Parser;
use std::{collections::HashMap, path::PathBuf};
use tracing::{debug, info, warn};
use tracing_subscriber::EnvFilter;

use lambda_code_examples::actions::{
    InvokeArgs::{Arithmetic, Increment},
    LambdaManager, Operation,
};

#[derive(Debug, Parser)]
pub struct Opt {
    /// The AWS Region.
    #[structopt(short, long)]
    pub region: Option<String>,

    // The bucket to use for the FunctionCode.
    #[structopt(short, long)]
    pub bucket: Option<String>,

    // The name of the Lambda function.
    #[structopt(short, long)]
    pub lambda_name: Option<String>,

    // The number to increment.
    #[structopt(short, long, default_value = "12")]
    pub inc: i32,

    // The left operand.
    #[structopt(long, default_value = "19")]
    pub num_a: i32,

    // The right operand.
    #[structopt(long, default_value = "23")]
    pub num_b: i32,

    // The arithmetic operation.
    #[structopt(short, long, default_value = "plus")]
    pub operation: Operation,

    #[structopt(long)]
```

```
    pub cleanup: Option<bool>,

    #[structopt(long)]
    pub no_cleanup: Option<bool>,
}

fn code_path(lambda: &str) -> PathBuf {
    PathBuf::from(format!("../target/lambda/{lambda}/bootstrap.zip"))
}

fn log_invoke_output(invoke: &InvokeOutput, message: &str) {
    if let Some(payload) = invoke.payload().cloned() {
        let payload = String::from_utf8(payload.into_inner());
        info!(?payload, message);
    } else {
        info!("Could not extract payload")
    }
    if let Some(logs) = invoke.log_result() {
        debug!(?logs, "Invoked function logs")
    } else {
        debug!("Invoked function had no logs")
    }
}

async fn main_block(
    opt: &Opt,
    manager: &LambdaManager,
    code_location: String,
) -> Result<(), anyhow::Error> {
    let invoke = manager.invoke(Increment(opt.inc)).await?;
    log_invoke_output(&invoke, "Invoked function configured as increment");

    let update_code = manager
        .update_function_code(code_path("arithmetic"), code_location.clone())
        .await?;

    let code_sha256 = update_code.code_sha256().unwrap_or("Unknown SHA");
    info!(?code_sha256, "Updated function code with arithmetic.zip");

    let arithmetic_args = Arithmetic(opt.operation, opt.num_a, opt.num_b);
    let invoke = manager.invoke(arithmetic_args).await?;
    log_invoke_output(&invoke, "Invoked function configured as arithmetic");

    let update = manager
```

```
        .update_function_configuration(
            Environment::builder()
                .set_variables(Some(HashMap::from([(
                    "RUST_LOG".to_string(),
                    "trace".to_string(),
                )])))
                .build(),
        )
        .await?;
    let updated_environment = update.environment();
    info!(?updated_environment, "Updated function configuration");

    let invoke = manager
        .invoke(Arithmetic(opt.operation, opt.num_a, opt.num_b))
        .await?;
    log_invoke_output(
        &invoke,
        "Invoked function configured as arithmetic with increased logging",
    );

    let invoke = manager
        .invoke(Arithmetic(Operation::DividedBy, opt.num_a, 0))
        .await?;
    log_invoke_output(
        &invoke,
        "Invoked function configured as arithmetic with divide by zero",
    );

    Ok::<(), anyhow::Error>(())
}

#[tokio::main]
async fn main() {
    tracing_subscriber::fmt()
        .without_time()
        .with_file(true)
        .with_line_number(true)
        .with_env_filter(EnvFilter::from_default_env())
        .init();

    let opt = Opt::parse();
    let manager = LambdaManager::load_from_env(opt.lambda_name.clone(),
 opt.bucket.clone()).await;
```

```
    let key = match manager.create_function(code_path("increment")).await {
        Ok(init) => {
            info!(?init, "Created function, initially with increment.zip");
            let run_block = main_block(&opt, &manager, init.clone()).await;
            info!(?run_block, "Finished running example, cleaning up");
            Some(init)
        }
        Err(err) => {
            warn!(?err, "Error happened when initializing function");
            None
        }
    };

    if Some(false) == opt.cleanup || Some(true) == opt.no_cleanup {
        info!("Skipping cleanup")
    } else {
        let delete = manager.cleanup(key).await;
        info!(?delete, "Deleted function & cleaned up resources");
    }
}
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK for Rust.
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Memanggil
  - ListFunctions
  - UpdateFunctionCode
  - UpdateFunctionConfiguration

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   TRY.
       "Create an AWS Identity and Access Management (IAM) role that grants AWS
Lambda permission to write to logs."
       DATA(lv_policy_document) = `{` &&
           `"Version":"2012-10-17",       ` &&
                 `"Statement": [` &&
                   `{` &&
                     `"Effect": "Allow",` &&
                     `"Action": [` &&
                       `"sts:AssumeRole"` &&
                     `],` &&
                     `"Principal": {` &&
                       `"Service": [` &&
                         `"lambda.amazonaws.com"` &&
                       `]` &&
                     `}` &&
                   `}` &&
                 `]` &&
               `}`.
       TRY.
           DATA(lo_create_role_output) = lo_iam->createrole(
                   iv_rolename = iv_role_name
                   iv_assumerolepolicydocument = lv_policy_document
                   iv_description = 'Grant lambda permission to write to
logs' ).
           DATA(lv_role_arn) = lo_create_role_output->get_role( )->get_arn( ).
           MESSAGE 'IAM role created.' TYPE 'I'.
           WAIT UP TO 10 SECONDS.           " Make sure that the IAM role is
ready for use. "
         CATCH /aws1/cx_iamentityalrdyexex.
           DATA(lo_role) = lo_iam->getrole( iv_rolename = iv_role_name ).
```

```
            lv_role_arn = lo_role->get_role( )->get_arn( ).
          CATCH /aws1/cx_iaminvalidinputex.
            MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
          CATCH /aws1/cx_iammalformedplydocex.
            MESSAGE 'Policy document in the request is malformed.' TYPE 'E'.
        ENDTRY.

        TRY.
            lo_iam->attachrolepolicy(
                iv_rolename  = iv_role_name
                iv_policyarn = 'arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole' ).
            MESSAGE 'Attached policy to the IAM role.' TYPE 'I'.
          CATCH /aws1/cx_iaminvalidinputex.
            MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
          CATCH /aws1/cx_iamnosuchentityex.
            MESSAGE 'The requested resource entity does not exist.' TYPE 'E'.
          CATCH /aws1/cx_iamplynotattachableex.
            MESSAGE 'Service role policies can only be attached to the service-
linked role for their service.' TYPE 'E'.
          CATCH /aws1/cx_iamunmodableentityex.
            MESSAGE 'Service that depends on the service-linked role is not
 modifiable.' TYPE 'E'.
        ENDTRY.

        " Create a Lambda function and upload handler code. "
        " Lambda function performs 'increment' action on a number. "
        TRY.
            lo_lmd->createfunction(
                 iv_functionname = iv_function_name
                 iv_runtime = `python3.9`
                 iv_role = lv_role_arn
                 iv_handler = iv_handler
                 io_code = io_initial_zip_file
                 iv_description = 'AWS Lambda code example' ).
            MESSAGE 'Lambda function created.' TYPE 'I'.
          CATCH /aws1/cx_lmdcodestorageexcdex.
            MESSAGE 'Maximum total code size per account exceeded.' TYPE 'E'.
          CATCH /aws1/cx_lmdinvparamvalueex.
            MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
          CATCH /aws1/cx_lmdresourcenotfoundex.
            MESSAGE 'The requested resource does not exist.' TYPE 'E'.
        ENDTRY.
```

```
        " Verify the function is in Active state "
        WHILE lo_lmd->getfunction( iv_functionname = iv_function_name )-
>get_configuration( )->ask_state( ) <> 'Active'.
          IF sy-index = 10.
            EXIT.               " Maximum 10 seconds. "
          ENDIF.
          WAIT UP TO 1 SECONDS.
        ENDWHILE.

        "Invoke the function with a single parameter and get results."
        TRY.
            DATA(lv_json) = /aws1/cl_rt_util=>string_to_xstring(
              `{`  &&
                `"action": "increment",`  &&
                `"number": 10` &&
              `}` ).
            DATA(lo_initial_invoke_output) = lo_lmd->invoke(
                       iv_functionname = iv_function_name
                       iv_payload = lv_json ).
            ov_initial_invoke_payload = lo_initial_invoke_output->get_payload( ).
           " ov_initial_invoke_payload is returned for testing purposes. "
            DATA(lo_writer_json) = cl_sxml_string_writer=>create( type =
 if_sxml=>co_xt_json ).
            CALL TRANSFORMATION id SOURCE XML ov_initial_invoke_payload RESULT
 XML lo_writer_json.
            DATA(lv_result) = cl_abap_codepage=>convert_from( lo_writer_json-
>get_output( ) ).
            MESSAGE 'Lambda function invoked.' TYPE 'I'.
          CATCH /aws1/cx_lmdinvparamvalueex.
            MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
          CATCH /aws1/cx_lmdinvrequestcontex.
            MESSAGE 'Unable to parse request body as JSON.' TYPE 'E'.
          CATCH /aws1/cx_lmdresourcenotfoundex.
            MESSAGE 'The requested resource does not exist.' TYPE 'E'.
          CATCH /aws1/cx_lmdunsuppedmediatyp00.
            MESSAGE 'Invoke request body does not have JSON as its content type.'
 TYPE 'E'.
        ENDTRY.

        " Update the function code and configure its Lambda environment with an
 environment variable. "
        " Lambda function is updated to perform 'decrement' action also. "
        TRY.
            lo_lmd->updatefunctioncode(
```

```
                iv_functionname = iv_function_name
                iv_zipfile = io_updated_zip_file ).
          WAIT UP TO 10 SECONDS.           " Make sure that the update is
 completed. "
          MESSAGE 'Lambda function code updated.' TYPE 'I'.
        CATCH /aws1/cx_lmdcodestorageexcdex.
          MESSAGE 'Maximum total code size per account exceeded.' TYPE 'E'.
        CATCH /aws1/cx_lmdinvparamvalueex.
          MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
        CATCH /aws1/cx_lmdresourcenotfoundex.
          MESSAGE 'The requested resource does not exist.' TYPE 'E'.
      ENDTRY.

      TRY.
          DATA lt_variables TYPE /aws1/
cl_lmdenvironmentvaria00=>tt_environmentvariables.
          DATA ls_variable LIKE LINE OF lt_variables.
          ls_variable-key = 'LOG_LEVEL'.
          ls_variable-value = NEW /aws1/cl_lmdenvironmentvaria00( iv_value =
 'info' ).
          INSERT ls_variable INTO TABLE lt_variables.

          lo_lmd->updatefunctionconfiguration(
                iv_functionname = iv_function_name
                io_environment = NEW /aws1/cl_lmdenvironment( it_variables =
 lt_variables ) ).
          WAIT UP TO 10 SECONDS.           " Make sure that the update is
 completed. "
          MESSAGE 'Lambda function configuration/settings updated.' TYPE 'I'.
        CATCH /aws1/cx_lmdinvparamvalueex.
          MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
        CATCH /aws1/cx_lmdresourceconflictex.
          MESSAGE 'Resource already exists or another operation is in
 progress.' TYPE 'E'.
        CATCH /aws1/cx_lmdresourcenotfoundex.
          MESSAGE 'The requested resource does not exist.' TYPE 'E'.
      ENDTRY.

      "Invoke the function with new parameters and get results. Display the
 execution log that's returned from the invocation."
      TRY.
          lv_json = /aws1/cl_rt_util=>string_to_xstring(
            `{`  &&
              `"action": "decrement",`  &&
```

```
                `"number": 10` &&
              `}` ).
            DATA(lo_updated_invoke_output) = lo_lmd->invoke(
                       iv_functionname = iv_function_name
                       iv_payload = lv_json ).
            ov_updated_invoke_payload = lo_updated_invoke_output->get_payload( ).
           " ov_updated_invoke_payload is returned for testing purposes. "
            lo_writer_json = cl_sxml_string_writer=>create( type =
 if_sxml=>co_xt_json ).
            CALL TRANSFORMATION id SOURCE XML ov_updated_invoke_payload RESULT
 XML lo_writer_json.
            lv_result = cl_abap_codepage=>convert_from( lo_writer_json-
>get_output( ) ).
            MESSAGE 'Lambda function invoked.' TYPE 'I'.
          CATCH /aws1/cx_lmdinvparamvalueex.
            MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
          CATCH /aws1/cx_lmdinvrequestcontex.
            MESSAGE 'Unable to parse request body as JSON.' TYPE 'E'.
          CATCH /aws1/cx_lmdresourcenotfoundex.
            MESSAGE 'The requested resource does not exist.' TYPE 'E'.
          CATCH /aws1/cx_lmdunsuppedmediatyp00.
            MESSAGE 'Invoke request body does not have JSON as its content type.'
 TYPE 'E'.
        ENDTRY.

        " List the functions for your account. "
        TRY.
            DATA(lo_list_output) = lo_lmd->listfunctions( ).
            DATA(lt_functions) = lo_list_output->get_functions( ).
            MESSAGE 'Retrieved list of Lambda functions.' TYPE 'I'.
          CATCH /aws1/cx_lmdinvparamvalueex.
            MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
        ENDTRY.

        " Delete the Lambda function. "
        TRY.
            lo_lmd->deletefunction( iv_functionname = iv_function_name ).
            MESSAGE 'Lambda function deleted.' TYPE 'I'.
          CATCH /aws1/cx_lmdinvparamvalueex.
            MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
          CATCH /aws1/cx_lmdresourcenotfoundex.
            MESSAGE 'The requested resource does not exist.' TYPE 'W'.
        ENDTRY.
```

```
        " Detach role policy. "
        TRY.
            lo_iam->detachrolepolicy(
                iv_rolename  = iv_role_name
                iv_policyarn = 'arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole' ).
            MESSAGE 'Detached policy from the IAM role.' TYPE 'I'.
          CATCH /aws1/cx_iaminvalidinputex.
            MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
          CATCH /aws1/cx_iamnosuchentityex.
            MESSAGE 'The requested resource entity does not exist.' TYPE 'W'.
          CATCH /aws1/cx_iamplynotattachableex.
            MESSAGE 'Service role policies can only be attached to the service-
linked role for their service.' TYPE 'E'.
          CATCH /aws1/cx_iamunmodableentityex.
            MESSAGE 'Service that depends on the service-linked role is not
 modifiable.' TYPE 'E'.
        ENDTRY.

        " Delete the IAM role. "
        TRY.
            lo_iam->deleterole( iv_rolename = iv_role_name ).
            MESSAGE 'IAM role deleted.' TYPE 'I'.
          CATCH /aws1/cx_iamnosuchentityex.
            MESSAGE 'The requested resource entity does not exist.' TYPE 'W'.
          CATCH /aws1/cx_iamunmodableentityex.
            MESSAGE 'Service that depends on the service-linked role is not
 modifiable.' TYPE 'E'.
        ENDTRY.

      CATCH /aws1/cx_rt_service_generic INTO lo_exception.
        DATA(lv_error) = lo_exception->get_longtext( ).
        MESSAGE lv_error TYPE 'E'.
    ENDTRY.
```

- Untuk mengetahui hal detail mengenai API, silakan lihat topik-topik berikut di referensi API AWS SDK untuk ABAP SAP.
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Memanggil

- ListFunctions
- UpdateFunctionCode
- UpdateFunctionConfiguration

Swift

SDK para Swift

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Tentukan fungsi Lambda pertama, yang hanya menambah nilai yang ditentukan.

```
// swift-tools-version: 5.9
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
//
// The swift-tools-version declares the minimum version of Swift required to
// build this package.

import PackageDescription

let package = Package(
    name: "increment",
    // Let Xcode know the minimum Apple platforms supported.
    platforms: [
        .macOS(.v13)
    ],
    dependencies: [
        // Dependencies declare other packages that this package depends on.
        .package(
            url: "https://github.com/swift-server/swift-aws-lambda-runtime.git",
            branch: "main"),
    ],
    targets: [
        // Targets are the basic building blocks of a package, defining a module
 or a test suite.
        // Targets can depend on other targets in this package and products
```

```
        // from dependencies.
        .executableTarget(
            name: "increment",
            dependencies: [
                .product(name: "AWSLambdaRuntime", package: "swift-aws-lambda-
runtime"),
            ],
            path: "Sources"
        )
    ]
)

import Foundation
import AWSLambdaRuntime

/// Represents the contents of the requests being received from the client.
/// This structure must be `Decodable` to indicate that its initializer
/// converts an external representation into this type.
struct Request: Decodable, Sendable {
    /// The action to perform.
    let action: String
    /// The number to act upon.
    let number: Int
}

/// The contents of the response sent back to the client. This must be
/// `Encodable`.
struct Response: Encodable, Sendable {
    /// The resulting value after performing the action.
    let answer: Int?
}


/// The Lambda function body.
///
/// - Parameters:
///   - event: The `Request` describing the request made by the
///     client.
///   - context: A `LambdaContext` describing the context in
///     which the lambda function is running.
///
/// - Returns: A `Response` object that will be encoded to JSON and sent
///   to the client by the Lambda runtime.
let incrementLambdaRuntime = LambdaRuntime {
```

```
        (event: Request, context: LambdaContext) -> Response in
    let action = event.action
    var answer: Int?

    if action != "increment" {
        context.logger.error("Unrecognized operation: \"\(action)\". The only
 supported action is \"increment\".")
    } else {
        answer = event.number + 1
        context.logger.info("The calculated answer is \(answer!).")
    }

    let response = Response(answer: answer)
    return response
}

// Run the Lambda runtime code.

try await incrementLambdaRuntime.run()
```

Tentukan fungsi Lambda kedua, yang melakukan operasi aritmatika pada dua angka.

```
// swift-tools-version: 5.9
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
//
// The swift-tools-version declares the minimum version of Swift required to
// build this package.

import PackageDescription

let package = Package(
    name: "calculator",
    // Let Xcode know the minimum Apple platforms supported.
    platforms: [
        .macOS(.v13)
    ],
    dependencies: [
        // Dependencies declare other packages that this package depends on.
        .package(
            url: "https://github.com/swift-server/swift-aws-lambda-runtime.git",
            branch: "main"),
```

```
    ],
    targets: [
        // Targets are the basic building blocks of a package, defining a module
 or a test suite.
        // Targets can depend on other targets in this package and products
        // from dependencies.
        .executableTarget(
            name: "calculator",
            dependencies: [
                .product(name: "AWSLambdaRuntime", package: "swift-aws-lambda-
runtime"),
            ],
            path: "Sources"
        )
    ]
)

import Foundation
import AWSLambdaRuntime

/// Represents the contents of the requests being received from the client.
/// This structure must be `Decodable` to indicate that its initializer
/// converts an external representation into this type.
struct Request: Decodable, Sendable {
    /// The action to perform.
    let action: String
    /// The first number to act upon.
    let x: Int
    /// The second number to act upon.
    let y: Int
}

/// A dictionary mapping operation names to closures that perform that
/// operation and return the result.
let actions = [
    "plus": { (x: Int, y: Int) -> Int in
        return x + y
    },
    "minus": { (x: Int, y: Int) -> Int in
        return x - y
    },
    "times": { (x: Int, y: Int) -> Int in
        return x * y
    },
```

```
    "divided-by": { (x: Int, y: Int) -> Int in
        return x / y
    }
]

/// The contents of the response sent back to the client. This must be
/// `Encodable`.
struct Response: Encodable, Sendable {
    /// The resulting value after performing the action.
    let answer: Int?
}


/// The Lambda function's entry point. Called by the Lambda runtime.
///
/// - Parameters:
///   - event: The `Request` describing the request made by the
///     client.
///   - context: A `LambdaContext` describing the context in
///     which the lambda function is running.
///
/// - Returns: A `Response` object that will be encoded to JSON and sent
///   to the client by the Lambda runtime.
let calculatorLambdaRuntime = LambdaRuntime {
        (_ event: Request, context: LambdaContext) -> Response in
    let action = event.action
    var answer: Int?
    var actionFunc: ((Int, Int) -> Int)?

    // Get the closure to run to perform the calculation.

    actionFunc = await actions[action]

    guard let actionFunc else {
        context.logger.error("Unrecognized operation '\(action)\'")
        return Response(answer: nil)
    }

    // Perform the calculation and return the answer.

    answer = actionFunc(event.x, event.y)

    guard let answer else {
        context.logger.error("Error computing \(event.x) \(action) \(event.y)")
```

```
    }
    context.logger.info("\(event.x) \(action) \(event.y) = \(answer)")

    return Response(answer: answer)
}

try await calculatorLambdaRuntime.run()
```

Tentukan program utama yang akan memanggil dua fungsi Lambda.

```
// swift-tools-version: 5.9
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
//
// The swift-tools-version declares the minimum version of Swift required to
// build this package.

import PackageDescription

let package = Package(
    name: "lambda-basics",
    // Let Xcode know the minimum Apple platforms supported.
    platforms: [
        .macOS(.v13)
    ],
    dependencies: [
        // Dependencies declare other packages that this package depends on.
        .package(
            url: "https://github.com/awslabs/aws-sdk-swift",
            from: "1.0.0"),
        .package(
            url: "https://github.com/apple/swift-argument-parser.git",
            branch: "main"
        )
    ],
    targets: [
        // Targets are the basic building blocks of a package, defining a module
 or a test suite.
        // Targets can depend on other targets in this package and products
        // from dependencies.
        .executableTarget(
            name: "lambda-basics",
```

```
            dependencies: [
                .product(name: "AWSLambda", package: "aws-sdk-swift"),
                .product(name: "AWSIAM", package: "aws-sdk-swift"),
                .product(name: "ArgumentParser", package: "swift-argument-
parser")
            ],
            path: "Sources"
        )
    ]
)

//
/// An example demonstrating a variety of important AWS Lambda functions.

import ArgumentParser
import AWSIAM
import SmithyWaitersAPI
import AWSClientRuntime
import AWSLambda
import Foundation

/// Represents the contents of the requests being received from the client.
/// This structure must be `Decodable` to indicate that its initializer
/// converts an external representation into this type.
struct IncrementRequest: Encodable, Decodable, Sendable {
    /// The action to perform.
    let action: String
    /// The number to act upon.
    let number: Int
}

struct Response: Encodable, Decodable, Sendable {
    /// The resulting value after performing the action.
    let answer: Int?
}

struct CalculatorRequest: Encodable, Decodable, Sendable {
    /// The action to perform.
    let action: String
    /// The first number to act upon.
    let x: Int
    /// The second number to act upon.
    let y: Int
}
```

```
let exampleName = "SwiftLambdaRoleExample"
let basicsFunctionName = "lambda-basics-function"

/// The ARN of the standard IAM policy for execution of Lambda functions.
let policyARN = "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole"

struct ExampleCommand: ParsableCommand {
    // -MARK: Command arguments
    @Option(help: "Name of the IAM Role to use for the Lambda functions")
    var role = exampleName
    @Option(help: "Zip archive containing the 'increment' lambda function")
    var incpath: String
    @Option(help: "Zip archive containing the 'calculator' lambda function")
    var calcpath: String
    @Option(help: "Name of the Amazon S3 Region to use (default: us-east-1)")
    var region = "us-east-1"

    static var configuration = CommandConfiguration(
        commandName: "lambda-basics",
        abstract: """
        This example demonstrates several common operations using AWS Lambda.
        """,
        discussion: """
        """
    )

    /// Returns the specified IAM role object.
    ///
    /// - Parameters:
    ///   - iamClient: `IAMClient` to use when looking for the role.
    ///   - roleName: The name of the role to check.
    ///
    /// - Returns: The `IAMClientTypes.Role` representing the specified role.
    func getRole(iamClient: IAMClient, roleName: String) async throws
                 -> IAMClientTypes.Role {
        do {
            let roleOutput = try await iamClient.getRole(
                input: GetRoleInput(
                    roleName: roleName
                )
            )
```

```
            guard let role = roleOutput.role else {
                throw ExampleError.roleNotFound
            }
            return role
        } catch {
            throw ExampleError.roleNotFound
        }
    }

    /// Create the AWS IAM role that will be used to access AWS Lambda.
    ///
    /// - Parameters:
    ///   - iamClient: The AWS `IAMClient` to use.
    ///   - roleName: The name of the AWS IAM role to use for Lambda.
    ///
    /// - Throws: `ExampleError.roleCreateError`
    ///
    /// - Returns: The `IAMClientTypes.Role` struct that describes the new role.
    func createRoleForLambda(iamClient: IAMClient, roleName: String) async throws
-> IAMClientTypes.Role {
        let output = try await iamClient.createRole(
            input: CreateRoleInput(
                assumeRolePolicyDocument:
                """
                {
                    "Version":"2012-10-17",
                    "Statement": [
                        {
                            "Effect": "Allow",
                            "Principal": {"Service": "lambda.amazonaws.com"},
                            "Action": "sts:AssumeRole"
                        }
                    ]
                }
                """,
                roleName: roleName
            )
        )

        guard let role = output.role else {
            throw ExampleError.roleCreateError
        }

        // Wait for the role to be ready for use.
```

```
        _ = try await iamClient.waitUntilRoleExists(
            options: WaiterOptions(
                maxWaitTime: 20,
                minDelay: 0.5,
                maxDelay: 2
            ),
            input: GetRoleInput(roleName: roleName)
        )

        return role
    }

    /// Detect whether or not the AWS Lambda function with the specified name
    /// exists, by requesting its function information.
    ///
    /// - Parameters:
    ///   - lambdaClient: The `LambdaClient` to use.
    ///   - name: The name of the AWS Lambda function to find.
    ///
    /// - Returns: `true` if the Lambda function exists. Otherwise `false`.
    func doesLambdaFunctionExist(lambdaClient: LambdaClient, name: String) async
-> Bool {
        do {
            _ = try await lambdaClient.getFunction(
                input: GetFunctionInput(functionName: name)
            )
        } catch {
            return false
        }

        return true
    }

    /// Create the specified AWS Lambda function.
    ///
    /// - Parameters:
    ///   - lambdaClient: The `LambdaClient` to use.
    ///   - functionName: The name of the AWS Lambda function to create.
    ///   - roleArn: The ARN of the role to apply to the function.
    ///   - path: The path of the Zip archive containing the function.
    ///
    /// - Returns: `true` if the AWS Lambda was successfully created; `false`
    ///   if it wasn't.
```

```
    func createFunction(lambdaClient: LambdaClient, functionName: String,
                                roleArn: String?, path: String) async throws ->
Bool {
        do {
            // Read the Zip archive containing the AWS Lambda function.

            let zipUrl = URL(fileURLWithPath: path)
            let zipData = try Data(contentsOf: zipUrl)

            // Create the AWS Lambda function that runs the specified code,
            // using the name given on the command line. The Lambda function
            // will run using the Amazon Linux 2 runtime.

            _ = try await lambdaClient.createFunction(
                input: CreateFunctionInput(
                    code: LambdaClientTypes.FunctionCode(zipFile: zipData),
                    functionName: functionName,
                    handler: "handle",
                    role: roleArn,
                    runtime: .providedal2
                )
            )
        } catch {
            print("*** Error creating Lambda function:")
            dump(error)
            return false
        }

        // Wait for a while to be sure the function is done being created.

        let output = try await lambdaClient.waitUntilFunctionActiveV2(
            options: WaiterOptions(
                maxWaitTime: 20,
                minDelay: 0.5,
                maxDelay: 2
            ),
            input: GetFunctionInput(functionName: functionName)
        )

        switch output.result {
            case .success:
                return true
            case .failure:
                return false
```

```
        }
    }

    /// Update the AWS Lambda function with new code to run when the function
    /// is invoked.
    ///
    /// - Parameters:
    ///   - lambdaClient: The `LambdaClient` to use.
    ///   - functionName: The name of the AWS Lambda function to update.
    ///   - path: The pathname of the Zip file containing the packaged Lambda
    ///     function.
    /// - Throws: `ExampleError.zipFileReadError`
    /// - Returns: `true` if the function's code is updated successfully.
    ///   Otherwise, returns `false`.
    func updateFunctionCode(lambdaClient: LambdaClient, functionName: String,
                            path: String) async throws -> Bool {
        let zipUrl = URL(fileURLWithPath: path)
        let zipData: Data

        // Read the function's Zip file.

        do {
            zipData = try Data(contentsOf: zipUrl)
        } catch {
            throw ExampleError.zipFileReadError
        }

        // Update the function's code and wait for the updated version to be
        // ready for use.

        do {
            _ = try await lambdaClient.updateFunctionCode(
                input: UpdateFunctionCodeInput(
                    functionName: functionName,
                    zipFile: zipData
                )
            )
        } catch {
            return false
        }

        let output = try await lambdaClient.waitUntilFunctionUpdatedV2(
            options: WaiterOptions(
                maxWaitTime: 20,
```

```
                minDelay: 0.5,
                maxDelay: 2
            ),
            input: GetFunctionInput(
                functionName: functionName
            )
        )

        switch output.result {
            case .success:
                return true
            case .failure:
                return false
        }
    }

    /// Tell the server-side component to log debug output by setting its
    /// environment's `LOG_LEVEL` to `DEBUG`.
    ///
    /// - Parameters:
    ///   - lambdaClient: The `LambdaClient` to use.
    ///   - functionName: The name of the AWS Lambda function to enable debug
    ///     logging for.
    ///
    /// - Throws: `ExampleError.environmentResponseMissingError`,
    ///   `ExampleError.updateFunctionConfigurationError`,
    ///   `ExampleError.environmentVariablesMissingError`,
    ///   `ExampleError.logLevelIncorrectError`,
    ///   `ExampleError.updateFunctionConfigurationError`
    func enableDebugLogging(lambdaClient: LambdaClient, functionName: String)
async throws {
        let envVariables = [
            "LOG_LEVEL": "DEBUG"
        ]
        let environment = LambdaClientTypes.Environment(variables: envVariables)

        do {
            let output = try await lambdaClient.updateFunctionConfiguration(
                input: UpdateFunctionConfigurationInput(
                    environment: environment,
                    functionName: functionName
                )
            )
```

```
            guard let response = output.environment else {
                throw ExampleError.environmentResponseMissingError
            }

            if response.error != nil {
                throw ExampleError.updateFunctionConfigurationError
            }

            guard let retVariables = response.variables else {
                throw ExampleError.environmentVariablesMissingError
            }

            for envVar in retVariables {
                if envVar.key == "LOG_LEVEL" && envVar.value != "DEBUG" {
                    print("*** Log level is not set to DEBUG!")
                    throw ExampleError.logLevelIncorrectError
                }
            }
        } catch {
            throw ExampleError.updateFunctionConfigurationError
        }
    }

    /// Returns an array containing the names of all AWS Lambda functions
    /// available to the user.
    ///
    /// - Parameter lambdaClient: The `IAMClient` to use.
    ///
    /// - Throws: `ExampleError.listFunctionsError`
    ///
    /// - Returns: An array of lambda function name strings.
    func getFunctionNames(lambdaClient: LambdaClient) async throws -> [String] {
        let pages = lambdaClient.listFunctionsPaginated(
            input: ListFunctionsInput()
        )

        var functionNames: [String] = []

        for try await page in pages {
            guard let functions = page.functions else {
                throw ExampleError.listFunctionsError
            }

            for function in functions {
```

```
                functionNames.append(function.functionName ?? "<unknown>")
            }
        }

        return functionNames
    }

    /// Invoke the Lambda function to increment a value.
    ///
    /// - Parameters:
    ///   - lambdaClient: The `IAMClient` to use.
    ///   - number: The number to increment.
    ///
    /// - Throws: `ExampleError.noAnswerReceived`, `ExampleError.invokeError`
    ///
    /// - Returns: An integer number containing the incremented value.
    func invokeIncrement(lambdaClient: LambdaClient, number: Int) async throws ->
Int {
        do {
            let incRequest = IncrementRequest(action: "increment", number:
number)
            let incData = try! JSONEncoder().encode(incRequest)

            // Invoke the lambda function.

            let invokeOutput = try await lambdaClient.invoke(
                input: InvokeInput(
                    functionName: "lambda-basics-function",
                    payload: incData
                )
            )

            let response = try! JSONDecoder().decode(Response.self,
from:invokeOutput.payload!)

            guard let answer = response.answer else {
                throw ExampleError.noAnswerReceived
            }
            return answer

        } catch {
            throw ExampleError.invokeError
        }
    }
```

```
    /// Invoke the calculator Lambda function.
    ///
    /// - Parameters:
    ///   - lambdaClient: The `IAMClient` to use.
    ///   - action: Which arithmetic operation to perform: "plus", "minus",
    ///     "times", or "divided-by".
    ///   - x: The first number to use in the computation.
    ///   - y: The second number to use in the computation.
    ///
    /// - Throws: `ExampleError.noAnswerReceived`, `ExampleError.invokeError`
    ///
    /// - Returns: The computed answer as an `Int`.
    func invokeCalculator(lambdaClient: LambdaClient, action: String, x: Int, y:
Int) async throws -> Int {
        do {
            let calcRequest = CalculatorRequest(action: action, x: x, y: y)
            let calcData = try! JSONEncoder().encode(calcRequest)

            // Invoke the lambda function.

            let invokeOutput = try await lambdaClient.invoke(
                input: InvokeInput(
                    functionName: "lambda-basics-function",
                    payload: calcData
                )
            )

            let response = try! JSONDecoder().decode(Response.self,
from:invokeOutput.payload!)

            guard let answer = response.answer else {
                throw ExampleError.noAnswerReceived
            }
            return answer

        } catch {
            throw ExampleError.invokeError
        }

    }

    /// Perform the example's tasks.
    func basics() async throws {
```

```
        let iamClient = try await IAMClient(
            config: IAMClient.IAMClientConfig(region: region)
        )

        let lambdaClient = try await LambdaClient(
            config: LambdaClient.LambdaClientConfig(region: region)
        )

        /// The IAM role to use for the example.
        var iamRole: IAMClientTypes.Role

        // Look for the specified role. If it already exists, use it. If not,
        // create it and attach the desired policy to it.

        do {
            iamRole = try await getRole(iamClient: iamClient, roleName: role)
        } catch ExampleError.roleNotFound {
            // The role wasn't found, so create it and attach the needed
            // policy.

            iamRole = try await createRoleForLambda(iamClient: iamClient,
roleName: role)

            do {
                _ = try await iamClient.attachRolePolicy(
                    input: AttachRolePolicyInput(policyArn: policyARN, roleName:
role)
                )
            } catch {
                throw ExampleError.policyError
            }
        }

        // Give the policy time to attach to the role.

        sleep(5)

        // Look to see if the function already exists. If it does, throw an
        // error.

        if await doesLambdaFunctionExist(lambdaClient: lambdaClient, name:
basicsFunctionName) {
            throw ExampleError.functionAlreadyExists
        }
```

```
        // Create, then invoke, the "increment" version of the calculator
        // function.

        print("Creating the increment Lambda function...")
        if try await createFunction(lambdaClient: lambdaClient, functionName:
basicsFunctionName,
                                    roleArn: iamRole.arn, path: incpath) {
            print("Running increment function calls...")
            for number in 0...4 {
                do {
                    let answer = try await invokeIncrement(lambdaClient:
lambdaClient, number: number)
                    print("Increment \(number) = \(answer)")
                } catch {
                    print("Error incrementing \(number): ",
error.localizedDescription)
                }
            }
        } else {
            print("*** Failed to create the increment function.")
        }

        // Enable debug logging.

        print("\nEnabling debug logging...")
        try await enableDebugLogging(lambdaClient: lambdaClient, functionName:
basicsFunctionName)

        // Change it to a basic arithmetic calculator. Then invoke it a few
        // times.

        print("\nReplacing the Lambda function with a calculator...")

        if try await updateFunctionCode(lambdaClient: lambdaClient, functionName:
basicsFunctionName,
                                        path: calcpath) {
            print("Running calculator function calls...")
            for x in [6, 10] {
                for y in [2, 4] {
                    for action in ["plus", "minus", "times", "divided-by"] {
                        do {
                            let answer = try await invokeCalculator(lambdaClient:
lambdaClient, action: action, x: x, y: y)
```

```
                            print("\(x) \(action) \(y) = \(answer)")
                        } catch {
                            print("Error calculating \(x) \(action) \(y): ",
error.localizedDescription)
                        }
                    }
                }
            }
        }

        // List all lambda functions.

        let functionNames = try await getFunctionNames(lambdaClient:
lambdaClient)

        if functionNames.count > 0 {
            print("\nAWS Lambda functions available on your account:")
            for name in functionNames {
                print("  \(name)")
            }
        }

        // Delete the lambda function.

        print("Deleting lambda function...")

        do {
            _ = try await lambdaClient.deleteFunction(
                input: DeleteFunctionInput(
                    functionName: "lambda-basics-function"
                )
            )
        } catch {
            print("Error: Unable to delete the function.")
        }

        // Detach the role from the policy, then delete the role.

        print("Deleting the AWS IAM role...")

        do {
            _ = try await iamClient.detachRolePolicy(
                input: DetachRolePolicyInput(
                    policyArn: policyARN,
```

```
                    roleName: role
                )
            )
            _ = try await iamClient.deleteRole(
                input: DeleteRoleInput(
                    roleName: role
                )
            )
        } catch {
            throw ExampleError.deleteRoleError
        }
    }
}

// -MARK: - Entry point

/// The program's asynchronous entry point.
@main
struct Main {
    static func main() async {
        let args = Array(CommandLine.arguments.dropFirst())

        do {
            let command = try ExampleCommand.parse(args)
            try await command.basics()
        } catch {
            ExampleCommand.exit(withError: error)
        }
    }
}


/// Errors thrown by the example's functions.
enum ExampleError: Error {
    /// An AWS Lambda function with the specified name already exists.
    case functionAlreadyExists
    /// The specified role doesn't exist.
    case roleNotFound
    /// Unable to create the role.
    case roleCreateError
    /// Unable to delete the role.
    case deleteRoleError
    /// Unable to attach a policy to the role.
    case policyError
```

```
    /// Unable to get the executable directory.
    case executableNotFound
    /// An error occurred creating a lambda function.
    case createLambdaError
    /// An error occurred invoking the lambda function.
    case invokeError
    /// No answer received from the invocation.
    case noAnswerReceived
    /// Unable to list the AWS Lambda functions.
    case listFunctionsError
    /// Unable to update the AWS Lambda function.
    case updateFunctionError
    /// Unable to update the function configuration.
    case updateFunctionConfigurationError
    /// The environment response is missing after an
    /// UpdateEnvironmentConfiguration attempt.
    case environmentResponseMissingError
    /// The environment variables are missing from the EnvironmentResponse and
    /// no errors occurred.
    case environmentVariablesMissingError
    /// The log level is incorrect after attempting to set it.
    case logLevelIncorrectError
    /// Unable to load the AWS Lambda function's Zip file.
    case zipFileReadError

    var errorDescription: String? {
        switch self {
        case .functionAlreadyExists:
            return "An AWS Lambda function with that name already exists."
        case .roleNotFound:
            return "The specified role doesn't exist."
        case .deleteRoleError:
            return "Unable to delete the AWS IAM role."
        case .roleCreateError:
            return "Unable to create the specified role."
        case .policyError:
            return "An error occurred attaching the policy to the role."
        case .executableNotFound:
            return "Unable to find the executable program directory."
        case .createLambdaError:
            return "An error occurred creating a lambda function."
        case .invokeError:
            return "An error occurred invoking a lambda function."
        case .noAnswerReceived:
```

```
            return "No answer received from the lambda function."
        case .listFunctionsError:
            return "Unable to list the AWS Lambda functions."
        case .updateFunctionError:
            return "Unable to update the AWS lambda function."
        case .updateFunctionConfigurationError:
            return "Unable to update the AWS lambda function configuration."
        case .environmentResponseMissingError:
            return "The environment is missing from the response after updating
 the function configuration."
        case .environmentVariablesMissingError:
            return "While no error occurred, no environment variables were
 returned following function configuration."
        case .logLevelIncorrectError:
            return "The log level is incorrect after attempting to set it to
 DEBUG."
        case .zipFileReadError:
            return "Unable to read the AWS Lambda function."
        }
    }
}
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK untuk Swift.
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Memanggil
  - ListFunctions
  - UpdateFunctionCode
  - UpdateFunctionConfiguration

Untuk daftar lengkap AWS Panduan pengembang SDK dan contoh kode, lihat Menggunakan Lambda dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Tindakan untuk Lambda menggunakan AWS SDK

Contoh kode berikut menunjukkan cara melakukan tindakan Lambda individual dengan AWS SDK. Setiap contoh menyertakan tautan ke GitHub, di mana Anda dapat menemukan instruksi untuk mengatur dan menjalankan kode.

Kutipan ini memanggil API Lambda dan merupakan kutipan kode dari program yang lebih besar yang harus dijalankan dalam konteks. Anda dapat melihat tindakan dalam konteks di Skenario untuk Lambda menggunakan AWS SDK.

Contoh berikut hanya mencakup tindakan yang paling umum digunakan. Untuk daftar lengkapnya, lihat AWS Lambda Referensi API.

Contoh

- Gunakan CreateAlias dengan CLI
- Gunakan CreateFunction dengan AWS SDK atau CLI
- Gunakan DeleteAlias dengan CLI
- Gunakan DeleteFunction dengan AWS SDK atau CLI
- Gunakan DeleteFunctionConcurrency dengan CLI
- Gunakan DeleteProvisionedConcurrencyConfig dengan CLI
- Gunakan GetAccountSettings dengan CLI
- Gunakan GetAlias dengan CLI
- Gunakan GetFunction dengan AWS SDK atau CLI
- Gunakan GetFunctionConcurrency dengan CLI
- Gunakan GetFunctionConfiguration dengan CLI
- Gunakan GetPolicy dengan CLI
- Gunakan GetProvisionedConcurrencyConfig dengan CLI
- Gunakan Memanggil dengan AWS SDK atau CLI
- Gunakan ListFunctions dengan AWS SDK atau CLI
- Gunakan ListProvisionedConcurrencyConfigs dengan CLI
- Gunakan ListTags dengan CLI
- Gunakan ListVersionsByFunction dengan CLI
- Gunakan PublishVersion dengan CLI

- Gunakan PutFunctionConcurrency dengan CLI
- Gunakan PutProvisionedConcurrencyConfig dengan CLI
- Gunakan RemovePermission dengan CLI
- Gunakan TagResource dengan CLI
- Gunakan UntagResource dengan CLI
- Gunakan UpdateAlias dengan CLI
- Gunakan UpdateFunctionCode dengan AWS SDK atau CLI
- Gunakan UpdateFunctionConfiguration dengan AWS SDK atau CLI

Gunakan **`CreateAlias`** dengan CLI

Contoh kode berikut menunjukkan cara menggunakan `CreateAlias`.

CLI

AWS CLI

Untuk membuat alias untuk fungsi Lambda

Berikut ini `create-alias` contoh membuat alias bernama `LIVE` yang menunjuk ke versi 1 dari `my-function` Fungsi lambda.

```
aws lambda create-alias \
    --function-name my-function \
    --description "alias for live version of function" \
    --function-version 1 \
    --name LIVE
```

Output:

```
{
    "FunctionVersion": "1",
    "Name": "LIVE",
    "AliasArn": "arn:aws:lambda:us-west-2:123456789012:function:my-
function:LIVE",
    "RevisionId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
    "Description": "alias for live version of function"
}
```

Untuk informasi lebih lanjut, lihat Mengkonfigurasi AWS Alias Fungsi Lambda di AWS Panduan Pengembang Lambda.

- Untuk detail API, lihat CreateAlias di AWS CLI Referensi Perintah.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Contoh ini membuat Alias Lambda Baru untuk versi tertentu dan konfigurasi routing untuk menentukan persentase permintaan pemanggilan yang diterimanya.

```
New-LMAlias -FunctionName "MylambdaFunction123" -
RoutingConfig_AdditionalVersionWeight @{Name="1";Value="0.6} -Description "Alias
 for version 4" -FunctionVersion 4 -Name "PowershellAlias"
```

- Untuk detail API, lihat CreateAlias di Alat AWS untuk PowerShell Referensi Cmdlet (V4).

Alat untuk PowerShell V5

Contoh 1: Contoh ini membuat Alias Lambda Baru untuk versi tertentu dan konfigurasi routing untuk menentukan persentase permintaan pemanggilan yang diterimanya.

```
New-LMAlias -FunctionName "MylambdaFunction123" -
RoutingConfig_AdditionalVersionWeight @{Name="1";Value="0.6} -Description "Alias
 for version 4" -FunctionVersion 4 -Name "PowershellAlias"
```

- Untuk detail API, lihat CreateAlias di Alat AWS untuk PowerShell Referensi Cmdlet (V5).

Untuk daftar lengkap AWS Panduan pengembang SDK dan contoh kode, lihat Menggunakan Lambda dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`CreateFunction`** dengan AWS SDK atau CLI

Contoh kode berikut menunjukkan cara menggunakan `CreateFunction`.

Contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Anda dapat melihat tindakan ini dalam konteks dalam contoh kode berikut:

- Pelajari dasar-dasarnya

- Buat API rest dengan integrasi proxy fungsi
- Membuat dasbor pemantauan dengan nama fungsi sebagai variabel
- Membuat fungsi tanpa server pertama Anda
- Menggunakan variabel properti dalam dasbor pemantauan untuk memantau beberapa fungsi tanpa server

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
/// <summary>
/// Creates a new Lambda function.
/// </summary>
/// <param name="functionName">The name of the function.</param>
/// <param name="s3Bucket">The Amazon Simple Storage Service (Amazon S3)
/// bucket where the zip file containing the code is located.</param>
/// <param name="s3Key">The Amazon S3 key of the zip file.</param>
/// <param name="role">The Amazon Resource Name (ARN) of a role with the
/// appropriate Lambda permissions.</param>
/// <param name="handler">The name of the handler function.</param>
/// <returns>The Amazon Resource Name (ARN) of the newly created
/// Lambda function.</returns>
public async Task<string> CreateLambdaFunctionAsync(
    string functionName,
    string s3Bucket,
    string s3Key,
    string role,
    string handler)
{
    // Defines the location for the function code.
    // S3Bucket - The S3 bucket where the file containing
    //            the source code is stored.
    // S3Key    - The name of the file containing the code.
    var functionCode = new FunctionCode
```

```
        {
            S3Bucket = s3Bucket,
            S3Key = s3Key,
        };

        var createFunctionRequest = new CreateFunctionRequest
        {
            FunctionName = functionName,
            Description = "Created by the Lambda .NET API",
            Code = functionCode,
            Handler = handler,
            Runtime = Runtime.Dotnet6,
            Role = role,
        };

        var reponse = await
_lambdaService.CreateFunctionAsync(createFunctionRequest);
        return reponse.FunctionArn;
    }
```

- Untuk detail API, lihat CreateFunction di AWS SDK untuk .NET Referensi API.

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region in which the bucket was created
(overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::Lambda::LambdaClient client(clientConfig);

        Aws::Lambda::Model::CreateFunctionRequest request;
```

```
        request.SetFunctionName(LAMBDA_NAME);
        request.SetDescription(LAMBDA_DESCRIPTION); // Optional.
#if USE_CPP_LAMBDA_FUNCTION
        request.SetRuntime(Aws::Lambda::Model::Runtime::provided_al2);
        request.SetTimeout(15);
        request.SetMemorySize(128);

        // Assume the AWS Lambda function was built in Docker with same
 architecture
        // as this code.
#if  defined(__x86_64__)
        request.SetArchitectures({Aws::Lambda::Model::Architecture::x86_64});
#elif defined(__aarch64__)
        request.SetArchitectures({Aws::Lambda::Model::Architecture::arm64});
#else
#error "Unimplemented architecture"
#endif // defined(architecture)
#else
        request.SetRuntime(Aws::Lambda::Model::Runtime::python3_9);
#endif
        request.SetRole(roleArn);
        request.SetHandler(LAMBDA_HANDLER_NAME);
        request.SetPublish(true);
        Aws::Lambda::Model::FunctionCode code;
        std::ifstream ifstream(INCREMENT_LAMBDA_CODE.c_str(),
                               std::ios_base::in | std::ios_base::binary);
        if (!ifstream.is_open()) {
            std::cerr << "Error opening file " << INCREMENT_LAMBDA_CODE << "." <<
 std::endl;

#if USE_CPP_LAMBDA_FUNCTION
            std::cerr
                    << "The cpp Lambda function must be built following the
 instructions in the cpp_lambda/README.md file. "
                    << std::endl;
#endif
            deleteIamRole(clientConfig);
            return false;
        }

        Aws::StringStream buffer;
        buffer << ifstream.rdbuf();
```

```
        code.SetZipFile(Aws::Utils::ByteBuffer((unsigned char *)
buffer.str().c_str(),
                                               buffer.str().length()));
        request.SetCode(code);

        Aws::Lambda::Model::CreateFunctionOutcome outcome =
client.CreateFunction(
                request);

        if (outcome.IsSuccess()) {
            std::cout << "The lambda function was successfully created. " <<
seconds
                      << " seconds elapsed." << std::endl;
            break;
        }

        else {
            std::cerr << "Error with CreateFunction. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            deleteIamRole(clientConfig);
            return false;
        }
```

- Untuk detail API, lihat CreateFunction di AWS SDK untuk C++ Referensi API.

CLI

AWS CLI

Untuk membuat fungsi Lambda

Berikut ini `create-function` contoh membuat fungsi Lambda bernama `my-function`.

```
aws lambda create-function \
    --function-name my-function \
    --runtime nodejs22.x \
    --zip-file fileb://my-function.zip \
    --handler my-function.handler \
    --role arn:aws:iam::123456789012:role/service-role/MyTestFunction-role-
tges6bf4
```

Isi dari `my-function.zip`:

```
This file is a deployment package that contains your function code and any
 dependencies.
```

Output:

```
{
    "TracingConfig": {
        "Mode": "PassThrough"
    },
    "CodeSha256": "PFn4S+er27qk+UuZSTKEQfNKG/XNn7QJs90mJgq6oH8=",
    "FunctionName": "my-function",
    "CodeSize": 308,
    "RevisionId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
    "MemorySize": 128,
    "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-function",
    "Version": "$LATEST",
    "Role": "arn:aws:iam::123456789012:role/service-role/MyTestFunction-role-
zgur6bf4",
    "Timeout": 3,
    "LastModified": "2025-10-14T22:26:11.234+0000",
    "Handler": "my-function.handler",
    "Runtime": "nodejs22.x",
    "Description": ""
}
```

Untuk informasi lebih lanjut, lihat Konfigurasikan memori fungsi Lambda di AWS Panduan Pengembang Lambda.

- Untuk detail API, lihat CreateFunction di AWS CLI Referensi Perintah.

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
import (
 "bytes"
 "context"
 "encoding/json"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/lambda"
 "github.com/aws/aws-sdk-go-v2/service/lambda/types"
)

// FunctionWrapper encapsulates function actions used in the examples.
// It contains an AWS Lambda service client that is used to perform user actions.
type FunctionWrapper struct {
 LambdaClient *lambda.Client
}



// CreateFunction creates a new Lambda function from code contained in the
 zipPackage
// buffer. The specified handlerName must match the name of the file and function
// contained in the uploaded code. The role specified by iamRoleArn is assumed by
// Lambda and grants specific permissions.
// When the function already exists, types.StateActive is returned.
// When the function is created, a lambda.FunctionActiveV2Waiter is used to wait
 until the
// function is active.
func (wrapper FunctionWrapper) CreateFunction(ctx context.Context, functionName
 string, handlerName string,
 iamRoleArn *string, zipPackage *bytes.Buffer) types.State {
 var state types.State
 _, err := wrapper.LambdaClient.CreateFunction(ctx, &lambda.CreateFunctionInput{
  Code:         &types.FunctionCode{ZipFile: zipPackage.Bytes()},
  FunctionName: aws.String(functionName),
  Role:         iamRoleArn,
  Handler:      aws.String(handlerName),
  Publish:      true,
  Runtime:      types.RuntimePython39,
 })
```

```
 if err != nil {
  var resConflict *types.ResourceConflictException
  if errors.As(err, &resConflict) {
   log.Printf("Function %v already exists.\n", functionName)
   state = types.StateActive
  } else {
   log.Panicf("Couldn't create function %v. Here's why: %v\n", functionName, err)
  }
 } else {
  waiter := lambda.NewFunctionActiveV2Waiter(wrapper.LambdaClient)
  funcOutput, err := waiter.WaitForOutput(ctx, &lambda.GetFunctionInput{
   FunctionName: aws.String(functionName)}, 1*time.Minute)
  if err != nil {
   log.Panicf("Couldn't wait for function %v to be active. Here's why: %v\n",
functionName, err)
  } else {
   state = funcOutput.Configuration.State
  }
 }
 return state
}
```

- Untuk detail API, lihat CreateFunction di AWS SDK untuk Go Referensi API.

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /**
     * Creates a new Lambda function in AWS using the AWS Lambda Java API.
     *
     * @param awsLambda    the AWS Lambda client used to interact with the AWS
Lambda service
     * @param functionName the name of the Lambda function to create
```

```
     * @param key          the S3 key of the function code
     * @param bucketName   the name of the S3 bucket containing the function code
     * @param role         the IAM role to assign to the Lambda function
     * @param handler      the fully qualified class name of the function handler
     * @return the Amazon Resource Name (ARN) of the created Lambda function
     */
    public static String createLambdaFunction(LambdaClient awsLambda,
                                               String functionName,
                                               String key,
                                               String bucketName,
                                               String role,
                                               String handler) {

        try {
            LambdaWaiter waiter = awsLambda.waiter();
            FunctionCode code = FunctionCode.builder()
                .s3Key(key)
                .s3Bucket(bucketName)
                .build();

            CreateFunctionRequest functionRequest =
CreateFunctionRequest.builder()
                .functionName(functionName)
                .description("Created by the Lambda Java API")
                .code(code)
                .handler(handler)
                .runtime(Runtime.JAVA17)
                .role(role)
                .build();

            // Create a Lambda function using a waiter
            CreateFunctionResponse functionResponse =
awsLambda.createFunction(functionRequest);
            GetFunctionRequest getFunctionRequest = GetFunctionRequest.builder()
                .functionName(functionName)
                .build();
            WaiterResponse<GetFunctionResponse> waiterResponse =
waiter.waitUntilFunctionExists(getFunctionRequest);
            waiterResponse.matched().response().ifPresent(System.out::println);
            return functionResponse.functionArn();

        } catch (LambdaException e) {
            System.err.println(e.getMessage());
            System.exit(1);
```

```
        }
        return "";
    }
```

- Untuk detail API, lihat CreateFunction di AWS SDK for Java 2.x Referensi API.

JavaScript

SDK untuk JavaScript (V3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
const createFunction = async (funcName, roleArn) => {
  const client = new LambdaClient({});
  const code = await readFile(`${dirname}../functions/${funcName}.zip`);

  const command = new CreateFunctionCommand({
    Code: { ZipFile: code },
    FunctionName: funcName,
    Role: roleArn,
    Architectures: [Architecture.arm64],
    Handler: "index.handler", // Required when sending a .zip file
    PackageType: PackageType.Zip, // Required when sending a .zip file
    Runtime: Runtime.nodejs16x, // Required when sending a .zip file
  });

  return client.send(command);
};
```

- Untuk detail API, lihat CreateFunction di AWS SDK untuk JavaScript Referensi API.

Kotlin

SDK untuk Kotlin

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
suspend fun createNewFunction(
    myFunctionName: String,
    s3BucketName: String,
    myS3Key: String,
    myHandler: String,
    myRole: String,
): String? {
    val functionCode =
        FunctionCode {
            s3Bucket = s3BucketName
            s3Key = myS3Key
        }

    val request =
        CreateFunctionRequest {
            functionName = myFunctionName
            code = functionCode
            description = "Created by the Lambda Kotlin API"
            handler = myHandler
            role = myRole
            runtime = Runtime.Java17
        }

    LambdaClient { region = "us-east-1" }.use { awsLambda ->
        val functionResponse = awsLambda.createFunction(request)
        awsLambda.waitUntilFunctionActive {
            functionName = myFunctionName
        }
        return functionResponse.functionArn
    }
}
```

- Untuk detail API, lihat CreateFunction di AWS SDK untuk referensi API Kotlin.

PHP

SDK untuk PHP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    public function createFunction($functionName, $role, $bucketName, $handler)
    {
        //This assumes the Lambda function is in an S3 bucket.
        return $this->customWaiter(function () use ($functionName, $role,
$bucketName, $handler) {
            return $this->lambdaClient->createFunction([
                'Code' => [
                    'S3Bucket' => $bucketName,
                    'S3Key' => $functionName,
                ],
                'FunctionName' => $functionName,
                'Role' => $role['Arn'],
                'Runtime' => 'python3.9',
                'Handler' => "$handler.lambda_handler",
            ]);
        });
    }
```

- Untuk detail API, lihat CreateFunction di AWS SDK untuk PHP Referensi API.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Contoh ini membuat fungsi C# (dotnetcore1.0 runtime) baru MyFunction bernamadiL AWS ambda, menyediakan binari terkompilasi untuk fungsi dari file zip pada sistem file lokal (jalur relatif atau absolut dapat digunakan). Fungsi C# Lambda menentukan handler untuk

fungsi menggunakan penunjukan AssemblyName::Namespace.ClassName::MethodName. Anda harus mengganti nama rakitan (tanpa akhiran .dll), namespace, nama kelas, dan bagian nama metode dari spesifikasi handler dengan tepat. Fungsi baru akan memiliki variabel lingkungan 'envvar1' dan 'envvar2' yang diatur dari nilai yang disediakan.

```
Publish-LMFunction -Description "My C# Lambda Function" `
        -FunctionName MyFunction `
        -ZipFilename .\MyFunctionBinaries.zip `
        -Handler "AssemblyName::Namespace.ClassName::MethodName" `
        -Role "arn:aws:iam::123456789012:role/LambdaFullExecRole" `
        -Runtime dotnetcore1.0 `
        -Environment_Variable @{ "envvar1"="value";"envvar2"="value" }
```

Output:

```
CodeSha256       : /NgBMd...gq71I=
CodeSize         : 214784
DeadLetterConfig :
Description      : My C# Lambda Function
Environment      : Amazon.Lambda.Model.EnvironmentResponse
FunctionArn      : arn:aws:lambda:us-west-2:123456789012:function:ToUpper
FunctionName     : MyFunction
Handler          : AssemblyName::Namespace.ClassName::MethodName
KMSKeyArn        :
LastModified     : 2016-12-29T23:50:14.207+0000
MemorySize       : 128
Role             : arn:aws:iam::123456789012:role/LambdaFullExecRole
Runtime          : dotnetcore1.0
Timeout          : 3
Version          : $LATEST
VpcConfig        :
```

Contoh 2: Contoh ini mirip dengan yang sebelumnya kecuali binari fungsi pertama kali diunggah ke bucket Amazon S3 (yang harus berada di wilayah yang sama dengan fungsi Lambda yang dimaksud) dan objek S3 yang dihasilkan kemudian direferensikan saat membuat fungsi.

```
Write-S3Object -BucketName amzn-s3-demo-bucket -Key MyFunctionBinaries.zip -
File .\MyFunctionBinaries.zip
Publish-LMFunction -Description "My C# Lambda Function" `
        -FunctionName MyFunction `
```

```
        -BucketName amzn-s3-demo-bucket `
        -Key MyFunctionBinaries.zip `
        -Handler "AssemblyName::Namespace.ClassName::MethodName" `
        -Role "arn:aws:iam::123456789012:role/LambdaFullExecRole" `
        -Runtime dotnetcore1.0 `
        -Environment_Variable @{ "envvar1"="value";"envvar2"="value" }
```

- Untuk detail API, lihat CreateFunction di Alat AWS untuk PowerShell Referensi Cmdlet (V4).

Alat untuk PowerShell V5

Contoh 1: Contoh ini membuat fungsi C# (dotnetcore1.0 runtime) baru MyFunction bernamadiL AWS ambda, menyediakan binari terkompilasi untuk fungsi dari file zip pada sistem file lokal (jalur relatif atau absolut dapat digunakan). Fungsi C# Lambda menentukan handler untuk fungsi menggunakan penunjukan AssemblyName::Namespace.ClassName::MethodName. Anda harus mengganti nama rakitan (tanpa akhiran .dll), namespace, nama kelas, dan bagian nama metode dari spesifikasi handler dengan tepat. Fungsi baru akan memiliki variabel lingkungan 'envvar1' dan 'envvar2' yang diatur dari nilai yang disediakan.

```
Publish-LMFunction -Description "My C# Lambda Function" `
        -FunctionName MyFunction `
        -ZipFilename .\MyFunctionBinaries.zip `
        -Handler "AssemblyName::Namespace.ClassName::MethodName" `
        -Role "arn:aws:iam::123456789012:role/LambdaFullExecRole" `
        -Runtime dotnetcore1.0 `
        -Environment_Variable @{ "envvar1"="value";"envvar2"="value" }
```

Output:

```
CodeSha256       : /NgBMd...gq71I=
CodeSize         : 214784
DeadLetterConfig :
Description      : My C# Lambda Function
Environment      : Amazon.Lambda.Model.EnvironmentResponse
FunctionArn      : arn:aws:lambda:us-west-2:123456789012:function:ToUpper
FunctionName     : MyFunction
Handler          : AssemblyName::Namespace.ClassName::MethodName
KMSKeyArn        :
LastModified     : 2016-12-29T23:50:14.207+0000
MemorySize       : 128
Role             : arn:aws:iam::123456789012:role/LambdaFullExecRole
Runtime          : dotnetcore1.0
Timeout          : 3
```

```
Version           : $LATEST
VpcConfig         :
```

Contoh 2: Contoh ini mirip dengan yang sebelumnya kecuali binari fungsi pertama kali diunggah ke bucket Amazon S3 (yang harus berada di wilayah yang sama dengan fungsi Lambda yang dimaksud) dan objek S3 yang dihasilkan kemudian direferensikan saat membuat fungsi.

```
Write-S3Object -BucketName amzn-s3-demo-bucket -Key MyFunctionBinaries.zip -
File .\MyFunctionBinaries.zip
Publish-LMFunction -Description "My C# Lambda Function" `
        -FunctionName MyFunction `
        -BucketName amzn-s3-demo-bucket `
        -Key MyFunctionBinaries.zip `
        -Handler "AssemblyName::Namespace.ClassName::MethodName" `
        -Role "arn:aws:iam::123456789012:role/LambdaFullExecRole" `
        -Runtime dotnetcore1.0 `
        -Environment_Variable @{ "envvar1"="value";"envvar2"="value" }
```

- Untuk detail API, lihat CreateFunction di Alat AWS untuk PowerShell Referensi Cmdlet (V5).

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
class LambdaWrapper:
    def __init__(self, lambda_client, iam_resource):
        self.lambda_client = lambda_client
        self.iam_resource = iam_resource


    def create_function(
        self, function_name, handler_name, iam_role, deployment_package
    ):
```

```
        """
        Deploys a Lambda function.

        :param function_name: The name of the Lambda function.
        :param handler_name: The fully qualified name of the handler function.
This
                             must include the file name and the function name.
        :param iam_role: The IAM role to use for the function.
        :param deployment_package: The deployment package that contains the
function
                                   code in .zip format.
        :return: The Amazon Resource Name (ARN) of the newly created function.
        """
        try:
            response = self.lambda_client.create_function(
                FunctionName=function_name,
                Description="AWS Lambda doc example",
                Runtime="python3.9",
                Role=iam_role.arn,
                Handler=handler_name,
                Code={"ZipFile": deployment_package},
                Publish=True,
            )
            function_arn = response["FunctionArn"]
            waiter = self.lambda_client.get_waiter("function_active_v2")
            waiter.wait(FunctionName=function_name)
            logger.info(
                "Created function '%s' with ARN: '%s'.",
                function_name,
                response["FunctionArn"],
            )
        except ClientError:
            logger.error("Couldn't create function %s.", function_name)
            raise
        else:
            return function_arn
```

- Untuk detail API, lihat CreateFunction di AWS Referensi API SDK untuk Python (Boto3).

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
class LambdaWrapper
  attr_accessor :lambda_client, :cloudwatch_client, :iam_client

  def initialize
    @lambda_client = Aws::Lambda::Client.new
    @cloudwatch_client = Aws::CloudWatchLogs::Client.new(region: 'us-east-1')
    @iam_client = Aws::IAM::Client.new(region: 'us-east-1')
    @logger = Logger.new($stdout)
    @logger.level = Logger::WARN
  end

  # Deploys a Lambda function.
  #
  # @param function_name: The name of the Lambda function.
  # @param handler_name: The fully qualified name of the handler function.
  # @param role_arn: The IAM role to use for the function.
  # @param deployment_package: The deployment package that contains the function
 code in .zip format.
  # @return: The Amazon Resource Name (ARN) of the newly created function.
  def create_function(function_name, handler_name, role_arn, deployment_package)
    response = @lambda_client.create_function({
                                                role: role_arn.to_s,
                                                function_name: function_name,
                                                handler: handler_name,
                                                runtime: 'ruby2.7',
                                                code: {
                                                  zip_file: deployment_package
                                                },
                                                environment: {
                                                  variables: {
                                                    'LOG_LEVEL' => 'info'
                                                  }
```

```
                                                  }
                                                })
   @lambda_client.wait_until(:function_active_v2, { function_name:
function_name }) do |w|
     w.max_attempts = 5
     w.delay = 5
   end
   response
 rescue Aws::Lambda::Errors::ServiceException => e
   @logger.error("There was an error creating #{function_name}:\n #{e.message}")
 rescue Aws::Waiters::Errors::WaiterFailed => e
   @logger.error("Failed waiting for #{function_name} to activate:\n
#{e.message}")
 end
```

- Untuk detail API, lihat CreateFunction di AWS SDK untuk Ruby Referensi API.

Rust

SDK for Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   /**
    * Create a function, uploading from a zip file.
    */
   pub async fn create_function(&self, zip_file: PathBuf) -> Result<String,
anyhow::Error> {
       let code = self.prepare_function(zip_file, None).await?;

       let key = code.s3_key().unwrap().to_string();

       let role = self.create_role().await.map_err(|e| anyhow!(e))?;

       info!("Created iam role, waiting 15s for it to become active");
       tokio::time::sleep(Duration::from_secs(15)).await;
```

```
        info!("Creating lambda function {}", self.lambda_name);
        let _ = self
            .lambda_client
            .create_function()
            .function_name(self.lambda_name.clone())
            .code(code)
            .role(role.arn())
            .runtime(aws_sdk_lambda::types::Runtime::Providedal2)
            .handler("_unused")
            .send()
            .await
            .map_err(anyhow::Error::from)?;

        self.wait_for_function_ready().await?;

        self.lambda_client
            .publish_version()
            .function_name(self.lambda_name.clone())
            .send()
            .await?;

        Ok(key)
    }

    /**
     * Upload function code from a path to a zip file.
     * The zip file must have an AL2 Linux-compatible binary called `bootstrap`.
     * The easiest way to create such a zip is to use `cargo lambda build --
output-format Zip`.
     */
    async fn prepare_function(
        &self,
        zip_file: PathBuf,
        key: Option<String>,
    ) -> Result<FunctionCode, anyhow::Error> {
        let body = ByteStream::from_path(zip_file).await?;

        let key = key.unwrap_or_else(|| format!("{}_code", self.lambda_name));

        info!("Uploading function code to s3://{}/{}", self.bucket, key);
        let _ = self
            .s3_client
            .put_object()
            .bucket(self.bucket.clone())
```

```
            .key(key.clone())
            .body(body)
            .send()
            .await?;

        Ok(FunctionCode::builder()
            .s3_bucket(self.bucket.clone())
            .s3_key(key)
            .build())
    }
```

- Untuk detail API, lihat CreateFunction di AWS Referensi SDK untuk Rust API.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    TRY.
        lo_lmd->createfunction(
            iv_functionname = iv_function_name
            iv_runtime = `python3.9`
            iv_role = iv_role_arn
            iv_handler = iv_handler
            io_code = io_zip_file
            iv_description = 'AWS Lambda code example' ).
        MESSAGE 'Lambda function created.' TYPE 'I'.
      CATCH /aws1/cx_lmdcodesigningcfgno00.
        MESSAGE 'Code signing configuration does not exist.' TYPE 'E'.
      CATCH /aws1/cx_lmdcodestorageexcdex.
        MESSAGE 'Maximum total code size per account exceeded.' TYPE 'E'.
      CATCH /aws1/cx_lmdcodeverification00.
        MESSAGE 'Code signature failed one or more validation checks for
signature mismatch or expiration.' TYPE 'E'.
      CATCH /aws1/cx_lmdinvalidcodesigex.
        MESSAGE 'Code signature failed the integrity check.' TYPE 'E'.
```

```
      CATCH /aws1/cx_lmdinvparamvalueex.
        MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
      CATCH /aws1/cx_lmdresourceconflictex.
        MESSAGE 'Resource already exists or another operation is in progress.'
 TYPE 'E'.
      CATCH /aws1/cx_lmdresourcenotfoundex.
        MESSAGE 'The requested resource does not exist.' TYPE 'E'.
      CATCH /aws1/cx_lmdserviceexception.
        MESSAGE 'An internal problem was encountered by the AWS Lambda service.'
 TYPE 'E'.
      CATCH /aws1/cx_lmdtoomanyrequestsex.
        MESSAGE 'The maximum request throughput was reached.' TYPE 'E'.
    ENDTRY.
```

- Untuk detail API, lihat CreateFunction di AWS SDK untuk referensi API SAP ABAP.

Swift

SDK para Swift

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
import AWSClientRuntime
import AWSLambda
import Foundation

        do {
            // Read the Zip archive containing the AWS Lambda function.

            let zipUrl = URL(fileURLWithPath: path)
            let zipData = try Data(contentsOf: zipUrl)

            // Create the AWS Lambda function that runs the specified code,
            // using the name given on the command line. The Lambda function
            // will run using the Amazon Linux 2 runtime.

            _ = try await lambdaClient.createFunction(
```

```
                input: CreateFunctionInput(
                    code: LambdaClientTypes.FunctionCode(zipFile: zipData),
                    functionName: functionName,
                    handler: "handle",
                    role: roleArn,
                    runtime: .providedal2
                )
            )
        } catch {
            print("*** Error creating Lambda function:")
            dump(error)
            return false
        }
```

- Untuk detail API, lihat CreateFunction di AWS SDK untuk referensi API Swift.

Untuk daftar lengkap AWS Panduan pengembang SDK dan contoh kode, lihat Menggunakan Lambda dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`DeleteAlias`** dengan CLI

Contoh kode berikut menunjukkan cara menggunakan `DeleteAlias`.

CLI

AWS CLI

Untuk menghapus alias dari fungsi Lambda

Berikut ini `delete-alias` contoh menghapus alias bernama `LIVE` dari `my-function` Fungsi lambda.

```
aws lambda delete-alias \
    --function-name my-function \
    --name LIVE
```

Perintah ini tidak menghasilkan output.

Untuk informasi lebih lanjut, lihat Mengkonfigurasi AWS Alias Fungsi Lambda di AWS Panduan Pengembang Lambda.

- Untuk detail API, lihat DeleteAlias di AWS CLI Referensi Perintah.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Contoh ini menghapus fungsi Lambda Alias yang disebutkan dalam perintah.

```
Remove-LMAlias -FunctionName "MylambdaFunction123" -Name "NewAlias"
```

- Untuk detail API, lihat DeleteAlias di Alat AWS untuk PowerShell Referensi Cmdlet (V4).

Alat untuk PowerShell V5

Contoh 1: Contoh ini menghapus fungsi Lambda Alias yang disebutkan dalam perintah.

```
Remove-LMAlias -FunctionName "MylambdaFunction123" -Name "NewAlias"
```

- Untuk detail API, lihat DeleteAlias di Alat AWS untuk PowerShell Referensi Cmdlet (V5).

Untuk daftar lengkap AWS Panduan pengembang SDK dan contoh kode, lihat Menggunakan Lambda dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`DeleteFunction`** dengan AWS SDK atau CLI

Contoh kode berikut menunjukkan cara menggunakan `DeleteFunction`.

Contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Anda dapat melihat tindakan ini dalam konteks dalam contoh kode berikut:

- Pelajari dasar-dasarnya
- Buat API rest dengan integrasi proxy fungsi
- Membuat dasbor pemantauan dengan nama fungsi sebagai variabel
- Membuat fungsi tanpa server pertama Anda
- Menggunakan variabel properti dalam dasbor pemantauan untuk memantau beberapa fungsi tanpa server

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Delete an AWS Lambda function.
    /// </summary>
    /// <param name="functionName">The name of the Lambda function to
    /// delete.</param>
    /// <returns>A Boolean value that indicates the success of the action.</
returns>
    public async Task<bool> DeleteFunctionAsync(string functionName)
    {
        var request = new DeleteFunctionRequest
        {
            FunctionName = functionName,
        };

        var response = await _lambdaService.DeleteFunctionAsync(request);

        // A return value of NoContent means that the request was processed.
        // In this case, the function was deleted, and the return value
        // is intentionally blank.
        return response.HttpStatusCode == System.Net.HttpStatusCode.NoContent;
    }
```

- Untuk detail API, lihat DeleteFunction di AWS SDK untuk .NET Referensi API.

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region in which the bucket was created
(overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::Lambda::LambdaClient client(clientConfig);

    Aws::Lambda::Model::DeleteFunctionRequest request;
    request.SetFunctionName(LAMBDA_NAME);

    Aws::Lambda::Model::DeleteFunctionOutcome outcome = client.DeleteFunction(
            request);

    if (outcome.IsSuccess()) {
        std::cout << "The lambda function was successfully deleted." <<
std::endl;
    }
    else {
        std::cerr << "Error with Lambda::DeleteFunction. "
                  << outcome.GetError().GetMessage()
                  << std::endl;
    }
```

- Untuk detail API, lihat DeleteFunction di AWS SDK untuk C++ Referensi API.

CLI

AWS CLI

Contoh 1: Untuk menghapus fungsi Lambda dengan nama fungsi

Berikut ini `delete-function` contoh menghapus fungsi Lambda bernama `my-function` dengan menentukan nama fungsi.

```
aws lambda delete-function \
    --function-name my-function
```

Perintah ini tidak menghasilkan output.

Contoh 2: Untuk menghapus fungsi Lambda dengan fungsi ARN

Berikut ini `delete-function` contoh menghapus fungsi Lambda bernama `my-function` dengan menentukan ARN fungsi.

```
aws lambda delete-function \
    --function-name arn:aws:lambda:us-west-2:123456789012:function:my-function
```

Perintah ini tidak menghasilkan output.

Contoh 3: Untuk menghapus fungsi Lambda dengan fungsi parSIAL ARN

Berikut ini `delete-function` contoh menghapus fungsi Lambda bernama `my-function` dengan menentukan ARN parSIAL fungsi.

```
aws lambda delete-function \
    --function-name 123456789012:function:my-function
```

Perintah ini tidak menghasilkan output.

Untuk informasi lebih lanjut, lihat AWS Konfigurasi Fungsi Lambda di AWS Panduan Pengembang Lambda.

- Untuk detail API, lihat DeleteFunction di AWS CLI Referensi Perintah.

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
import (
 "bytes"
 "context"
 "encoding/json"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/lambda"
 "github.com/aws/aws-sdk-go-v2/service/lambda/types"
)

// FunctionWrapper encapsulates function actions used in the examples.
// It contains an AWS Lambda service client that is used to perform user actions.
type FunctionWrapper struct {
 LambdaClient *lambda.Client
}



// DeleteFunction deletes the Lambda function specified by functionName.
func (wrapper FunctionWrapper) DeleteFunction(ctx context.Context, functionName
 string) {
 _, err := wrapper.LambdaClient.DeleteFunction(ctx, &lambda.DeleteFunctionInput{
  FunctionName: aws.String(functionName),
 })
 if err != nil {
  log.Panicf("Couldn't delete function %v. Here's why: %v\n", functionName, err)
 }
}
```

- Untuk detail API, lihat DeleteFunction di AWS SDK untuk Go Referensi API.

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   /**
    * Deletes an AWS Lambda function.
    *
    * @param awsLambda     an instance of the {@link LambdaClient} class, which
is used to interact with the AWS Lambda service
    * @param functionName  the name of the Lambda function to be deleted
    *
    * @throws LambdaException if an error occurs while deleting the Lambda
function
    */
   public static void deleteLambdaFunction(LambdaClient awsLambda, String
functionName) {
       try {
           DeleteFunctionRequest request = DeleteFunctionRequest.builder()
               .functionName(functionName)
               .build();

           awsLambda.deleteFunction(request);
           System.out.println("The " + functionName + " function was deleted");

       } catch (LambdaException e) {
           System.err.println(e.getMessage());
           System.exit(1);
       }
   }
```

- Untuk detail API, lihat DeleteFunction di AWS SDK for Java 2.x Referensi API.

JavaScript

SDK untuk JavaScript (V3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
/**
 * @param {string} funcName
 */
const deleteFunction = (funcName) => {
  const client = new LambdaClient({});
  const command = new DeleteFunctionCommand({ FunctionName: funcName });
  return client.send(command);
};
```

- Untuk detail API, lihat DeleteFunction di AWS SDK untuk JavaScript Referensi API.

Kotlin

SDK untuk Kotlin

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
suspend fun delLambdaFunction(myFunctionName: String) {
    val request =
        DeleteFunctionRequest {
            functionName = myFunctionName
        }

    LambdaClient { region = "us-east-1" }.use { awsLambda ->
        awsLambda.deleteFunction(request)
```

```
        println("$myFunctionName was deleted")
    }
}
```

- Untuk detail API, lihat DeleteFunction di AWS SDK untuk referensi API Kotlin.

PHP

SDK untuk PHP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    public function deleteFunction($functionName)
    {
        return $this->lambdaClient->deleteFunction([
            'FunctionName' => $functionName,
        ]);
    }
```

- Untuk detail API, lihat DeleteFunction di AWS SDK untuk PHP Referensi API.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Contoh ini menghapus versi tertentu dari fungsi Lambda

```
Remove-LMFunction -FunctionName "MylambdaFunction123" -Qualifier '3'
```

- Untuk detail API, lihat DeleteFunction di Alat AWS untuk PowerShell Referensi Cmdlet (V4).

Alat untuk PowerShell V5

Contoh 1: Contoh ini menghapus versi tertentu dari fungsi Lambda

```
Remove-LMFunction -FunctionName "MylambdaFunction123" -Qualifier '3'
```

- Untuk detail API, lihat DeleteFunction di Alat AWS untuk PowerShell Referensi Cmdlet (V5).

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
class LambdaWrapper:
    def __init__(self, lambda_client, iam_resource):
        self.lambda_client = lambda_client
        self.iam_resource = iam_resource


    def delete_function(self, function_name):
        """
        Deletes a Lambda function.

        :param function_name: The name of the function to delete.
        """
        try:
            self.lambda_client.delete_function(FunctionName=function_name)
        except ClientError:
            logger.exception("Couldn't delete function %s.", function_name)
            raise
```

- Untuk detail API, lihat DeleteFunction di AWS Referensi API SDK untuk Python (Boto3).

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
class LambdaWrapper
  attr_accessor :lambda_client, :cloudwatch_client, :iam_client

  def initialize
    @lambda_client = Aws::Lambda::Client.new
    @cloudwatch_client = Aws::CloudWatchLogs::Client.new(region: 'us-east-1')
    @iam_client = Aws::IAM::Client.new(region: 'us-east-1')
    @logger = Logger.new($stdout)
    @logger.level = Logger::WARN
  end

  # Deletes a Lambda function.
  # @param function_name: The name of the function to delete.
  def delete_function(function_name)
    print "Deleting function: #{function_name}..."
    @lambda_client.delete_function(
      function_name: function_name
    )
    print 'Done!'.green
  rescue Aws::Lambda::Errors::ServiceException => e
    @logger.error("There was an error deleting #{function_name}:\n #{e.message}")
  end
```

- Untuk detail API, lihat DeleteFunction di AWS SDK untuk Ruby Referensi API.

Rust

SDK for Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   /** Delete a function and its role, and if possible or necessary, its
associated code object and bucket. */
   pub async fn delete_function(
       &self,
       location: Option<String>,
   ) -> (
       Result<DeleteFunctionOutput, anyhow::Error>,
       Result<DeleteRoleOutput, anyhow::Error>,
       Option<Result<DeleteObjectOutput, anyhow::Error>>,
   ) {
       info!("Deleting lambda function {}", self.lambda_name);
       let delete_function = self
           .lambda_client
           .delete_function()
           .function_name(self.lambda_name.clone())
           .send()
           .await
           .map_err(anyhow::Error::from);

       info!("Deleting iam role {}", self.role_name);
       let delete_role = self
           .iam_client
           .delete_role()
           .role_name(self.role_name.clone())
           .send()
           .await
           .map_err(anyhow::Error::from);

       let delete_object: Option<Result<DeleteObjectOutput, anyhow::Error>> =
           if let Some(location) = location {
               info!("Deleting object {location}");
               Some(
```

```
                    self.s3_client
                        .delete_object()
                        .bucket(self.bucket.clone())
                        .key(location)
                        .send()
                        .await
                        .map_err(anyhow::Error::from),
                )
            } else {
                info!(?location, "Skipping delete object");
                None
            };

        (delete_function, delete_role, delete_object)
    }
```

- Untuk detail API, lihat DeleteFunction di AWS Referensi SDK untuk Rust API.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    TRY.
        lo_lmd->deletefunction( iv_functionname = iv_function_name ).
        MESSAGE 'Lambda function deleted.' TYPE 'I'.
      CATCH /aws1/cx_lmdinvparamvalueex.
        MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
      CATCH /aws1/cx_lmdresourceconflictex.
        MESSAGE 'Resource already exists or another operation is in progress.'
TYPE 'E'.
      CATCH /aws1/cx_lmdresourcenotfoundex.
        MESSAGE 'The requested resource does not exist.' TYPE 'E'.
      CATCH /aws1/cx_lmdserviceexception.
        MESSAGE 'An internal problem was encountered by the AWS Lambda service.'
TYPE 'E'.
```

```
      CATCH /aws1/cx_lmdtoomanyrequestsex.
        MESSAGE 'The maximum request throughput was reached.' TYPE 'E'.
    ENDTRY.
```

- Untuk detail API, lihat DeleteFunction di AWS SDK untuk referensi API SAP ABAP.

Swift

SDK para Swift

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
import AWSClientRuntime
import AWSLambda
import Foundation

        do {
            _ = try await lambdaClient.deleteFunction(
                input: DeleteFunctionInput(
                    functionName: "lambda-basics-function"
                )
            )
        } catch {
            print("Error: Unable to delete the function.")
        }
```

- Untuk detail API, lihat DeleteFunction di AWS SDK untuk referensi API Swift.

Untuk daftar lengkap AWS Panduan pengembang SDK dan contoh kode, lihat Menggunakan Lambda dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`DeleteFunctionConcurrency`** dengan CLI

Contoh kode berikut menunjukkan cara menggunakan `DeleteFunctionConcurrency`.

CLI

AWS CLI

Untuk menghapus batas eksekusi bersamaan yang dicadangkan dari fungsi

Berikut ini `delete-function-concurrency` contoh menghapus batas eksekusi bersamaan yang dicadangkan dari `my-function` fungsi.

```
aws lambda delete-function-concurrency \
    --function-name  my-function
```

Perintah ini tidak menghasilkan output.

Untuk informasi lebih lanjut, lihat Memesan Konkurensi untuk Fungsi Lambda di AWS Panduan Pengembang Lambda.

- Untuk detail API, lihat DeleteFunctionConcurrency di AWS CLI Referensi Perintah.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Contoh ini menghapus Concurrency Fungsi dari Fungsi Lambda.

```
Remove-LMFunctionConcurrency -FunctionName "MylambdaFunction123"
```

- Untuk detail API, lihat DeleteFunctionConcurrency di Alat AWS untuk PowerShell Referensi Cmdlet (V4).

Alat untuk PowerShell V5

Contoh 1: Contoh ini menghapus Concurrency Fungsi dari Fungsi Lambda.

```
Remove-LMFunctionConcurrency -FunctionName "MylambdaFunction123"
```

- Untuk detail API, lihat DeleteFunctionConcurrency di Alat AWS untuk PowerShell Referensi Cmdlet (V5).

Untuk daftar lengkap AWS Panduan pengembang SDK dan contoh kode, lihat Menggunakan Lambda dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan `DeleteProvisionedConcurrencyConfig` dengan CLI

Contoh kode berikut menunjukkan cara menggunakan `DeleteProvisionedConcurrencyConfig`.

CLI

AWS CLI

Untuk menghapus konfigurasi konkurensi yang disediakan

Berikut ini `delete-provisioned-concurrency-config` contoh menghapus konfigurasi konkurensi yang disediakan untuk `GREEN` alias dari fungsi yang ditentukan.

```
aws lambda delete-provisioned-concurrency-config \
    --function-name my-function \
    --qualifier GREEN
```

Perintah ini tidak menghasilkan output.

Untuk informasi lebih lanjut, lihat Mengkonfigurasi konkurensi yang disediakan untuk Fungsi Peristiwa di AWS Panduan Pengembang Lambda.

- Untuk detail API, lihat DeleteProvisionedConcurrencyConfig di AWS CLI Referensi Perintah.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Contoh ini menghapus Konfigurasi Konkurensi yang Disediakan untuk Alias tertentu.

```
Remove-LMProvisionedConcurrencyConfig -FunctionName "MylambdaFunction123" -
Qualifier "NewAlias1"
```

- Untuk detail API, lihat DeleteProvisionedConcurrencyConfig di Alat AWS untuk PowerShell Referensi Cmdlet (V4).

Alat untuk PowerShell V5

Contoh 1: Contoh ini menghapus Konfigurasi Konkurensi yang Disediakan untuk Alias tertentu.

```
Remove-LMProvisionedConcurrencyConfig -FunctionName "MylambdaFunction123" -
Qualifier "NewAlias1"
```

- Untuk detail API, lihat DeleteProvisionedConcurrencyConfig di Alat AWS untuk PowerShell Referensi Cmdlet (V5).

Untuk daftar lengkap AWS Panduan pengembang SDK dan contoh kode, lihat Menggunakan Lambda dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`GetAccountSettings`** dengan CLI

Contoh kode berikut menunjukkan cara menggunakan `GetAccountSettings`.

CLI

AWS CLI

Untuk mengambil rincian tentang akun Anda di AWS Wilayah

Berikut ini `get-account-settings` contoh menampilkan batas Lambda dan informasi penggunaan untuk akun Anda.

```
aws lambda get-account-settings
```

Output:

```
{
    "AccountLimit": {
        "CodeSizeUnzipped": 262144000,
        "UnreservedConcurrentExecutions": 1000,
        "ConcurrentExecutions": 1000,
        "CodeSizeZipped": 52428800,
        "TotalCodeSize": 80530636800
    },
    "AccountUsage": {
        "FunctionCount": 4,
        "TotalCodeSize": 9426
    }
}
```

Untuk informasi lebih lanjut, lihat AWS Batas Lambda di AWS Panduan Pengembang Lambda.

- Untuk detail API, lihat GetAccountSettings di AWS CLI Referensi Perintah.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Contoh ini ditampilkan untuk membandingkan Batas Akun dan Penggunaan Akun

```
Get-LMAccountSetting | Select-Object
 @{Name="TotalCodeSizeLimit";Expression={$_.AccountLimit.TotalCodeSize}},
 @{Name="TotalCodeSizeUsed";Expression={$_.AccountUsage.TotalCodeSize}}
```

Output:

```
TotalCodeSizeLimit TotalCodeSizeUsed
------------------ -----------------
       80530636800          15078795
```

- Untuk detail API, lihat GetAccountSettings di Alat AWS untuk PowerShell Referensi Cmdlet (V4).

Alat untuk PowerShell V5

Contoh 1: Contoh ini ditampilkan untuk membandingkan Batas Akun dan Penggunaan Akun

```
Get-LMAccountSetting | Select-Object
 @{Name="TotalCodeSizeLimit";Expression={$_.AccountLimit.TotalCodeSize}},
 @{Name="TotalCodeSizeUsed";Expression={$_.AccountUsage.TotalCodeSize}}
```

Output:

```
TotalCodeSizeLimit TotalCodeSizeUsed
------------------ -----------------
       80530636800          15078795
```

- Untuk detail API, lihat GetAccountSettings di Alat AWS untuk PowerShell Referensi Cmdlet (V5).

Untuk daftar lengkap AWS Panduan pengembang SDK dan contoh kode, lihat Menggunakan Lambda dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`GetAlias`** dengan CLI

Contoh kode berikut menunjukkan cara menggunakan `GetAlias`.

CLI

AWS CLI

Untuk mengambil detail tentang alias fungsi

Berikut ini `get-alias` contoh menampilkan detail untuk nama alias `LIVE` pada `my-function` Fungsi lambda.

```
aws lambda get-alias \
    --function-name my-function \
    --name LIVE
```

Output:

```
{
    "FunctionVersion": "3",
    "Name": "LIVE",
    "AliasArn": "arn:aws:lambda:us-west-2:123456789012:function:my-
function:LIVE",
    "RevisionId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
    "Description": "alias for live version of function"
}
```

Untuk informasi lebih lanjut, lihat Mengkonfigurasi AWS Alias Fungsi Lambda di AWS Panduan Pengembang Lambda.

- Untuk detail API, lihat GetAlias di AWS CLI Referensi Perintah.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Contoh ini mengambil bobot Routing Config untuk Alias Fungsi Lambda tertentu.

```
Get-LMAlias -FunctionName "MylambdaFunction123" -Name "newlabel1" -Select
 RoutingConfig
```

Output:

```
AdditionalVersionWeights
------------------------
{[1, 0.6]}
```

- Untuk detail API, lihat GetAlias di Alat AWS untuk PowerShell Referensi Cmdlet (V4).

Alat untuk PowerShell V5

Contoh 1: Contoh ini mengambil bobot Routing Config untuk Alias Fungsi Lambda tertentu.

```
Get-LMAlias -FunctionName "MylambdaFunction123" -Name "newlabel1" -Select
 RoutingConfig
```

Output:

```
AdditionalVersionWeights
------------------------
{[1, 0.6]}
```

- Untuk detail API, lihat GetAlias di Alat AWS untuk PowerShell Referensi Cmdlet (V5).

Untuk daftar lengkap AWS Panduan pengembang SDK dan contoh kode, lihat Menggunakan Lambda dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`GetFunction`** dengan AWS SDK atau CLI

Contoh kode berikut menunjukkan cara menggunakan `GetFunction`.

Contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Anda dapat melihat tindakan ini dalam konteks dalam contoh kode berikut:

- Pelajari dasar-dasarnya
- Membuat fungsi tanpa server pertama Anda

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Gets information about a Lambda function.
    /// </summary>
    /// <param name="functionName">The name of the Lambda function for
    /// which to retrieve information.</param>
    /// <returns>Async Task.</returns>
    public async Task<FunctionConfiguration> GetFunctionAsync(string
functionName)
    {
        var functionRequest = new GetFunctionRequest
        {
            FunctionName = functionName,
        };

        var response = await _lambdaService.GetFunctionAsync(functionRequest);
        return response.Configuration;
    }
```

- Untuk detail API, lihat GetFunction di AWS SDK untuk .NET Referensi API.

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region in which the bucket was created
(overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::Lambda::LambdaClient client(clientConfig);

        Aws::Lambda::Model::GetFunctionRequest request;
        request.SetFunctionName(functionName);

        Aws::Lambda::Model::GetFunctionOutcome outcome =
client.GetFunction(request);

        if (outcome.IsSuccess()) {
            std::cout << "Function retrieve.\n" <<

outcome.GetResult().GetConfiguration().Jsonize().View().WriteReadable()
                      << std::endl;
        }
        else {
            std::cerr << "Error with Lambda::GetFunction. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
        }
```

- Untuk detail API, lihat GetFunction di AWS SDK untuk C++ Referensi API.

CLI

AWS CLI

Untuk mengambil informasi tentang suatu fungsi

Berikut ini `get-function` contoh menampilkan informasi tentang `my-function` fungsi.

```
aws lambda get-function \
    --function-name  my-function
```

Output:

```
{
```

```
    "Concurrency": {
        "ReservedConcurrentExecutions": 100
    },
    "Code": {
        "RepositoryType": "S3",
        "Location": "https://awslambda-us-west-2-tasks.s3.us-
west-2.amazonaws.com/snapshots/123456789012/my-function..."
    },
    "Configuration": {
        "TracingConfig": {
            "Mode": "PassThrough"
        },
        "Version": "$LATEST",
        "CodeSha256": "5tT2qgzYUHoqwR616pZ2dpkn/0J1FrzJmlKidWaaCgk=",
        "FunctionName": "my-function",
        "VpcConfig": {
            "SubnetIds": [],
            "VpcId": "",
            "SecurityGroupIds": []
        },
        "MemorySize": 128,
        "RevisionId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "CodeSize": 304,
        "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-
function",
        "Handler": "index.handler",
        "Role": "arn:aws:iam::123456789012:role/service-role/helloWorldPython-
role-uy3l9qyq",
        "Timeout": 3,
        "LastModified": "2025-09-24T18:20:35.054+0000",
        "Runtime": "nodejs22.x",
        "Description": ""
    }
}
```


Untuk informasi lebih lanjut, lihat Konfigurasikan memori fungsi Lambda di AWS Panduan Pengembang Lambda.

- Untuk detail API, lihat GetFunction di AWS CLI Referensi Perintah.

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
import (
 "bytes"
 "context"
 "encoding/json"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/lambda"
 "github.com/aws/aws-sdk-go-v2/service/lambda/types"
)

// FunctionWrapper encapsulates function actions used in the examples.
// It contains an AWS Lambda service client that is used to perform user actions.
type FunctionWrapper struct {
 LambdaClient *lambda.Client
}



// GetFunction gets data about the Lambda function specified by functionName.
func (wrapper FunctionWrapper) GetFunction(ctx context.Context, functionName
 string) types.State {
 var state types.State
 funcOutput, err := wrapper.LambdaClient.GetFunction(ctx,
 &lambda.GetFunctionInput{
  FunctionName: aws.String(functionName),
 })
 if err != nil {
  log.Panicf("Couldn't get function %v. Here's why: %v\n", functionName, err)
```

```
 } else {
  state = funcOutput.Configuration.State
 }
 return state
}
```

- Untuk detail API, lihat GetFunction di AWS SDK untuk Go Referensi API.

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /**
     * Retrieves information about an AWS Lambda function.
     *
     * @param awsLambda    an instance of the {@link LambdaClient} class, which
is used to interact with the AWS Lambda service
     * @param functionName the name of the AWS Lambda function to retrieve
information about
     */
    public static void getFunction(LambdaClient awsLambda, String functionName) {
        try {
            GetFunctionRequest functionRequest = GetFunctionRequest.builder()
                .functionName(functionName)
                .build();

            GetFunctionResponse response =
awsLambda.getFunction(functionRequest);
            System.out.println("The runtime of this Lambda function is " +
response.configuration().runtime());

        } catch (LambdaException e) {
            System.err.println(e.getMessage());
            System.exit(1);
```

```
        }
    }
```

- Untuk detail API, lihat GetFunction di AWS SDK for Java 2.x Referensi API.

JavaScript

SDK untuk JavaScript (V3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
const getFunction = (funcName) => {
  const client = new LambdaClient({});
  const command = new GetFunctionCommand({ FunctionName: funcName });
  return client.send(command);
};
```

- Untuk detail API, lihat GetFunction di AWS SDK untuk JavaScript Referensi API.

PHP

SDK untuk PHP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    public function getFunction($functionName)
    {
        return $this->lambdaClient->getFunction([
```

```
            'FunctionName' => $functionName,
        ]);
    }
```

- Untuk detail API, lihat GetFunction di AWS SDK untuk PHP Referensi API.

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
class LambdaWrapper:
    def __init__(self, lambda_client, iam_resource):
        self.lambda_client = lambda_client
        self.iam_resource = iam_resource


    def get_function(self, function_name):
        """
        Gets data about a Lambda function.

        :param function_name: The name of the function.
        :return: The function data.
        """
        response = None
        try:
            response =
 self.lambda_client.get_function(FunctionName=function_name)
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.info("Function %s does not exist.", function_name)
            else:
                logger.error(
                    "Couldn't get function %s. Here's why: %s: %s",
                    function_name,
                    err.response["Error"]["Code"],
```

```
                    err.response["Error"]["Message"],
                )
                raise
        return response
```

- Untuk detail API, lihat GetFunction di AWS Referensi API SDK untuk Python (Boto3).

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
class LambdaWrapper
  attr_accessor :lambda_client, :cloudwatch_client, :iam_client

  def initialize
    @lambda_client = Aws::Lambda::Client.new
    @cloudwatch_client = Aws::CloudWatchLogs::Client.new(region: 'us-east-1')
    @iam_client = Aws::IAM::Client.new(region: 'us-east-1')
    @logger = Logger.new($stdout)
    @logger.level = Logger::WARN
  end

  # Gets data about a Lambda function.
  #
  # @param function_name: The name of the function.
  # @return response: The function data, or nil if no such function exists.
  def get_function(function_name)
    @lambda_client.get_function(
      {
        function_name: function_name
      }
    )
  rescue Aws::Lambda::Errors::ResourceNotFoundException => e
    @logger.debug("Could not find function: #{function_name}:\n #{e.message}")
```

```
    nil
  end
```

- Untuk detail API, lihat GetFunction di AWS SDK untuk Ruby Referensi API.

Rust

SDK for Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /** Get the Lambda function with this Manager's name. */
    pub async fn get_function(&self) -> Result<GetFunctionOutput, anyhow::Error>
{
        info!("Getting lambda function");
        self.lambda_client
            .get_function()
            .function_name(self.lambda_name.clone())
            .send()
            .await
            .map_err(anyhow::Error::from)
    }
```

- Untuk detail API, lihat GetFunction di AWS Referensi SDK untuk Rust API.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    TRY.
        oo_result = lo_lmd->getfunction( iv_functionname = iv_function_name ).
    " oo_result is returned for testing purposes. "
        MESSAGE 'Lambda function information retrieved.' TYPE 'I'.
      CATCH /aws1/cx_lmdinvparamvalueex.
        MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
      CATCH /aws1/cx_lmdserviceexception.
        MESSAGE 'An internal problem was encountered by the AWS Lambda service.'
 TYPE 'E'.
      CATCH /aws1/cx_lmdtoomanyrequestsex.
        MESSAGE 'The maximum request throughput was reached.' TYPE 'E'.
    ENDTRY.
```

- Untuk detail API, lihat GetFunction di AWS SDK untuk referensi API SAP ABAP.

Swift

SDK para Swift

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
import AWSClientRuntime
import AWSLambda
import Foundation

    /// Detect whether or not the AWS Lambda function with the specified name
    /// exists, by requesting its function information.
    ///
    /// - Parameters:
    ///   - lambdaClient: The `LambdaClient` to use.
    ///   - name: The name of the AWS Lambda function to find.
    ///
    /// - Returns: `true` if the Lambda function exists. Otherwise `false`.
    func doesLambdaFunctionExist(lambdaClient: LambdaClient, name: String) async
 -> Bool {
        do {
```

```
            _ = try await lambdaClient.getFunction(
                input: GetFunctionInput(functionName: name)
            )
        } catch {
            return false
        }

        return true
    }
```

- Untuk detail API, lihat GetFunction di AWS SDK untuk referensi API Swift.

Untuk daftar lengkap AWS Panduan pengembang SDK dan contoh kode, lihat Menggunakan Lambda dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`GetFunctionConcurrency`** dengan CLI

Contoh kode berikut menunjukkan cara menggunakan `GetFunctionConcurrency`.

CLI

AWS CLI

Untuk melihat pengaturan konkurensi yang dicadangkan untuk suatu fungsi

Berikut ini `get-function-concurrency` contoh mengambil pengaturan konkurensi yang dicadangkan untuk fungsi yang ditentukan.

```
aws lambda get-function-concurrency \
    --function-name my-function
```

Output:

```
{
    "ReservedConcurrentExecutions": 250
}
```

Untuk informasi lebih lanjut, lihat Mengkonfigurasi konkurensi yang dicadangkan untuk Fungsi Peristiwa di AWS Panduan Pengembang Lambda.

- Untuk detail API, lihat GetFunctionConcurrency di AWS CLI Referensi Perintah.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Contoh ini mendapatkan konkurensi Cadangan untuk Fungsi Lambda

```
Get-LMFunctionConcurrency -FunctionName "MylambdaFunction123" -Select *
```

Output:

```
ReservedConcurrentExecutions
----------------------------
100
```

- Untuk detail API, lihat GetFunctionConcurrency di Alat AWS untuk PowerShell Referensi Cmdlet (V4).

Alat untuk PowerShell V5

Contoh 1: Contoh ini mendapatkan konkurensi Cadangan untuk Fungsi Lambda

```
Get-LMFunctionConcurrency -FunctionName "MylambdaFunction123" -Select *
```

Output:

```
ReservedConcurrentExecutions
----------------------------
100
```

- Untuk detail API, lihat GetFunctionConcurrency di Alat AWS untuk PowerShell Referensi Cmdlet (V5).

Untuk daftar lengkap AWS Panduan pengembang SDK dan contoh kode, lihat Menggunakan Lambda dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`GetFunctionConfiguration`** dengan CLI

Contoh kode berikut menunjukkan cara menggunakan `GetFunctionConfiguration`.

Contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Anda dapat melihat tindakan ini dalam konteks dalam contoh kode berikut:

- Membuat fungsi tanpa server pertama Anda

CLI

AWS CLI

Untuk mengambil pengaturan khusus versi dari fungsi Lambda

Berikut ini `get-function-configuration` contoh menampilkan pengaturan untuk versi 2 dari `my-function` fungsi.

```
aws lambda get-function-configuration \
    --function-name  my-function:2
```

Output:

```
{
    "FunctionName": "my-function",
    "LastModified": "2019-09-26T20:28:40.438+0000",
    "RevisionId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
    "MemorySize": 256,
    "Version": "2",
    "Role": "arn:aws:iam::123456789012:role/service-role/my-function-role-
uy3l9qyq",
    "Timeout": 3,
    "Runtime": "nodejs10.x",
    "TracingConfig": {
        "Mode": "PassThrough"
    },
    "CodeSha256": "5tT2qgzYUHaqwR716pZ2dpkn/0J1FrzJmlKidWoaCgk=",
    "Description": "",
    "VpcConfig": {
        "SubnetIds": [],
        "VpcId": "",
        "SecurityGroupIds": []
    },
    "CodeSize": 304,
    "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-
function:2",
```

```
    "Handler": "index.handler"
}
```

Untuk informasi lebih lanjut, lihat AWS Konfigurasi Fungsi Lambda di AWS Panduan Pengembang Lambda.

- Untuk detail API, lihat GetFunctionConfiguration di AWS CLI Referensi Perintah.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Contoh ini mengembalikan konfigurasi spesifik versi Fungsi Lambda.

```
Get-LMFunctionConfiguration -FunctionName "MylambdaFunction123" -Qualifier
 "PowershellAlias"
```

Output:

```
CodeSha256                  : uWOW0R7z+f0VyLuUg7+/D08hkMFsq0SF4seuyUZJ/R8=
CodeSize                    : 1426
DeadLetterConfig            : Amazon.Lambda.Model.DeadLetterConfig
Description                 : Verson 3 to test Aliases
Environment                 : Amazon.Lambda.Model.EnvironmentResponse
FunctionArn                 : arn:aws:lambda:us-
east-1:123456789012:function:MylambdaFunction123
                              :PowershellAlias
FunctionName                : MylambdaFunction123
Handler                     : lambda_function.launch_instance
KMSKeyArn                   :
LastModified                : 2019-12-25T09:52:59.872+0000
LastUpdateStatus            : Successful
LastUpdateStatusReason      :
LastUpdateStatusReasonCode  :
Layers                      : {}
MasterArn                   :
MemorySize                  : 128
RevisionId                  : 5d7de38b-87f2-4260-8f8a-e87280e10c33
Role                        : arn:aws:iam::123456789012:role/service-role/lambda
Runtime                     : python3.8
State                       : Active
StateReason                 :
```

```
StateReasonCode            :
Timeout                    : 600
TracingConfig              : Amazon.Lambda.Model.TracingConfigResponse
Version                    : 4
VpcConfig                  : Amazon.Lambda.Model.VpcConfigDetail
```

- Untuk detail API, lihat GetFunctionConfiguration di Alat AWS untuk PowerShell Referensi Cmdlet (V4).

Alat untuk PowerShell V5

Contoh 1: Contoh ini mengembalikan konfigurasi spesifik versi Fungsi Lambda.

```
Get-LMFunctionConfiguration -FunctionName "MylambdaFunction123" -Qualifier
 "PowershellAlias"
```

Output:

```
CodeSha256                 : uWOW0R7z+f0VyLuUg7+/D08hkMFsq0SF4seuyUZJ/R8=
CodeSize                   : 1426
DeadLetterConfig           : Amazon.Lambda.Model.DeadLetterConfig
Description                : Verson 3 to test Aliases
Environment                : Amazon.Lambda.Model.EnvironmentResponse
FunctionArn                : arn:aws:lambda:us-
east-1:123456789012:function:MylambdaFunction123
                             :PowershellAlias
FunctionName               : MylambdaFunction123
Handler                    : lambda_function.launch_instance
KMSKeyArn                  :
LastModified               : 2019-12-25T09:52:59.872+0000
LastUpdateStatus           : Successful
LastUpdateStatusReason     :
LastUpdateStatusReasonCode :
Layers                     : {}
MasterArn                  :
MemorySize                 : 128
RevisionId                 : 5d7de38b-87f2-4260-8f8a-e87280e10c33
Role                       : arn:aws:iam::123456789012:role/service-role/lambda
Runtime                    : python3.8
State                      : Active
StateReason                :
StateReasonCode            :
Timeout                    : 600
```

```
TracingConfig               : Amazon.Lambda.Model.TracingConfigResponse
Version                     : 4
VpcConfig                   : Amazon.Lambda.Model.VpcConfigDetail
```

- Untuk detail API, lihat GetFunctionConfiguration di Alat AWS untuk PowerShell Referensi Cmdlet (V5).

Untuk daftar lengkap AWS Panduan pengembang SDK dan contoh kode, lihat Menggunakan Lambda dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`GetPolicy`** dengan CLI

Contoh kode berikut menunjukkan cara menggunakan `GetPolicy`.

CLI

AWS CLI

Untuk mengambil kebijakan IAM berbasis sumber daya untuk fungsi, versi, atau alias

Berikut ini `get-policy` contoh menampilkan informasi kebijakan tentang `my-function` Fungsi lambda.

```
aws lambda get-policy \
    --function-name my-function
```

Output:

```
{
    "Policy": "{\"Version\":\"2012-10-17\",\"Id\":\"default\",\"Statement
\":[{\"Sid\":\"iot-events\",\"Effect\":\"Allow\",\"Principal\":{\"Service\":
\"iotevents.amazonaws.com\"},\"Action\":\"lambda:InvokeFunction\",\"Resource\":
\"arn:aws:lambda:us-west-2:123456789012:function:my-function\"}]}",
    "RevisionId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
}
```

Untuk informasi lebih lanjut, lihat Menggunakan Resource-based Kebijakan untuk AWS Lambda di AWS Panduan Pengembang Lambda.

- Untuk detail API, lihat GetPolicy di AWS CLI Referensi Perintah.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Sampel ini menampilkan kebijakan Function dari fungsi Lambda

```
Get-LMPolicy -FunctionName test -Select Policy
```

Output:

```
{"Version":"2012-10-17",       "Id":"default","Statement":
[{"Sid":"xxxx","Effect":"Allow","Principal":
{"Service":"sns.amazonaws.com"},"Action":"lambda:InvokeFunction","Resource":"arn:aws:lamb
east-1:123456789102:function:test"}]}
```

- Untuk detail API, lihat GetPolicy di Alat AWS untuk PowerShell Referensi Cmdlet (V4).

Alat untuk PowerShell V5

Contoh 1: Sampel ini menampilkan kebijakan Function dari fungsi Lambda

```
Get-LMPolicy -FunctionName test -Select Policy
```

Output:

```
{"Version":"2012-10-17",       "Id":"default","Statement":
[{"Sid":"xxxx","Effect":"Allow","Principal":
{"Service":"sns.amazonaws.com"},"Action":"lambda:InvokeFunction","Resource":"arn:aws:lamb
east-1:123456789102:function:test"}]}
```

- Untuk detail API, lihat GetPolicy di Alat AWS untuk PowerShell Referensi Cmdlet (V5).

Untuk daftar lengkap AWS Panduan pengembang SDK dan contoh kode, lihat Menggunakan Lambda dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`GetProvisionedConcurrencyConfig`** dengan CLI

Contoh kode berikut menunjukkan cara menggunakan `GetProvisionedConcurrencyConfig`.

CLI

AWS CLI

Untuk melihat konfigurasi konkurensi yang disediakan

Berikut ini `get-provisioned-concurrency-config` contoh menampilkan detail untuk konfigurasi konkurensi yang disediakan untuk `BLUE` alias dari fungsi yang ditentukan.

```
aws lambda get-provisioned-concurrency-config \
    --function-name my-function \
    --qualifier BLUE
```

Output:

```
{
    "RequestedProvisionedConcurrentExecutions": 100,
    "AvailableProvisionedConcurrentExecutions": 100,
    "AllocatedProvisionedConcurrentExecutions": 100,
    "Status": "READY",
    "LastModified": "2019-12-31T20:28:49+0000"
}
```

Untuk informasi lebih lanjut, lihat Mengkonfigurasi konkurensi yang disediakan untuk Fungsi Peristiwa di AWS Panduan Pengembang Lambda.

- Untuk detail API, lihat GetProvisionedConcurrencyConfig di AWS CLI Referensi Perintah.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Contoh ini mendapatkan Konfigurasi Konkurensi yang disediakan untuk Alias yang ditentukan dari Fungsi Lambda.

```
C:\>Get-LMProvisionedConcurrencyConfig -FunctionName "MylambdaFunction123" -
Qualifier "NewAlias1"
```

Output:

```
AllocatedProvisionedConcurrentExecutions : 0
```

```
AvailableProvisionedConcurrentExecutions : 0
LastModified                             : 2020-01-15T03:21:26+0000
RequestedProvisionedConcurrentExecutions : 70
Status                                   : IN_PROGRESS
StatusReason                             :
```

- Untuk detail API, lihat GetProvisionedConcurrencyConfig di Alat AWS untuk PowerShell Referensi Cmdlet (V4).

Alat untuk PowerShell V5

Contoh 1: Contoh ini mendapatkan Konfigurasi Konkurensi yang disediakan untuk Alias yang ditentukan dari Fungsi Lambda.

```
C:\>Get-LMProvisionedConcurrencyConfig -FunctionName "MylambdaFunction123" -
Qualifier "NewAlias1"
```

Output:

```
AllocatedProvisionedConcurrentExecutions : 0
AvailableProvisionedConcurrentExecutions : 0
LastModified                             : 2020-01-15T03:21:26+0000
RequestedProvisionedConcurrentExecutions : 70
Status                                   : IN_PROGRESS
StatusReason                             :
```

- Untuk detail API, lihat GetProvisionedConcurrencyConfig di Alat AWS untuk PowerShell Referensi Cmdlet (V5).

Untuk daftar lengkap AWS Panduan pengembang SDK dan contoh kode, lihat Menggunakan Lambda dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`Memanggil`** dengan AWS SDK atau CLI

Contoh kode berikut menunjukkan cara menggunakan `Invoke`.

Contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Anda dapat melihat tindakan ini dalam konteks dalam contoh kode berikut:

- Pelajari dasar-dasarnya

- Membuat dasbor pemantauan dengan nama fungsi sebagai variabel
- Membuat fungsi tanpa server pertama Anda
- Menggunakan variabel properti dalam dasbor pemantauan untuk memantau beberapa fungsi tanpa server

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Invoke a Lambda function.
    /// </summary>
    /// <param name="functionName">The name of the Lambda function to
    /// invoke.</param
    /// <param name="parameters">The parameter values that will be passed to the
function.</param>
    /// <returns>A System Threading Task.</returns>
    public async Task<string> InvokeFunctionAsync(
        string functionName,
        string parameters)
    {
        var payload = parameters;
        var request = new InvokeRequest
        {
            FunctionName = functionName,
            Payload = payload,
        };

        var response = await _lambdaService.InvokeAsync(request);
        MemoryStream stream = response.Payload;
        string returnValue =
System.Text.Encoding.UTF8.GetString(stream.ToArray());
        return returnValue;
    }
```

- Untuk detail API, lihat Memanggil di AWS SDK untuk .NET Referensi API.

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region in which the bucket was created
(overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::Lambda::LambdaClient client(clientConfig);

        Aws::Lambda::Model::InvokeRequest request;
        request.SetFunctionName(LAMBDA_NAME);
        request.SetLogType(logType);
        std::shared_ptr<Aws::IOStream> payload =
Aws::MakeShared<Aws::StringStream>(
                "FunctionTest");
        *payload << jsonPayload.View().WriteReadable();
        request.SetBody(payload);
        request.SetContentType("application/json");
        Aws::Lambda::Model::InvokeOutcome outcome = client.Invoke(request);

        if (outcome.IsSuccess()) {
            invokeResult = std::move(outcome.GetResult());
            result = true;
            break;
        }

        else {
            std::cerr << "Error with Lambda::InvokeRequest. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
```

```
            break;
        }
```

- Untuk detail API, lihat Memanggil di AWS SDK untuk C++ Referensi API.

CLI

AWS CLI

Contoh 1: Untuk memanggil fungsi Lambda secara sinkron

Berikut ini `invoke` contoh memanggil `my-function` berfungsi secara sinkron. `cli-binary-format` opsi diperlukan jika Anda menggunakan AWS CLI versi 2. Untuk informasi lebih lanjut, lihat AWS CLI mendukung opsi baris perintah global di AWS Panduan Pengguna Antarmuka Baris Perintah.

```
aws lambda invoke response.json \
    --function-name my-function \
    --cli-binary-format raw-in-base64-out \
    --payload '{ "name": "Bob" }'
```

Output:

```
{
    "ExecutedVersion": "$LATEST",
    "StatusCode": 200
}
```

Untuk informasi lebih lanjut, lihat Memanggil fungsi Lambda secara sinkron di AWS Panduan Pengembang Lambda.

Contoh 2: Untuk memanggil fungsi Lambda secara asinkron

Berikut ini `invoke` contoh memanggil `my-function` berfungsi secara asinkron. `cli-binary-format` opsi diperlukan jika Anda menggunakan AWS CLI versi 2. Untuk informasi lebih lanjut, lihat AWS CLI mendukung opsi baris perintah global di AWS Panduan Pengguna Antarmuka Baris Perintah.

```
aws lambda invoke response.json \
    --function-name my-function \
```

```
    --invocation-type Event \
    --cli-binary-format raw-in-base64-out \
    --payload '{ "name": "Bob" }'
```

Output:

```
{
    "StatusCode": 202
}
```

Untuk informasi lebih lanjut, lihat Memanggil fungsi Lambda secara asinkron di AWS Panduan Pengembang Lambda.

- Untuk detail API, lihat Memanggil di AWS CLI Referensi Perintah.

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
import (
 "bytes"
 "context"
 "encoding/json"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/lambda"
 "github.com/aws/aws-sdk-go-v2/service/lambda/types"
)

// FunctionWrapper encapsulates function actions used in the examples.
// It contains an AWS Lambda service client that is used to perform user actions.
```

```
type FunctionWrapper struct {
 LambdaClient *lambda.Client
}



// Invoke invokes the Lambda function specified by functionName, passing the
 parameters
// as a JSON payload. When getLog is true, types.LogTypeTail is specified, which
 tells
// Lambda to include the last few log lines in the returned result.
func (wrapper FunctionWrapper) Invoke(ctx context.Context, functionName string,
 parameters any, getLog bool) *lambda.InvokeOutput {
 logType := types.LogTypeNone
 if getLog {
  logType = types.LogTypeTail
 }
 payload, err := json.Marshal(parameters)
 if err != nil {
  log.Panicf("Couldn't marshal parameters to JSON. Here's why %v\n", err)
 }
 invokeOutput, err := wrapper.LambdaClient.Invoke(ctx, &lambda.InvokeInput{
  FunctionName: aws.String(functionName),
  LogType:      logType,
  Payload:      payload,
 })
 if err != nil {
  log.Panicf("Couldn't invoke function %v. Here's why: %v\n", functionName, err)
 }
 return invokeOutput
}
```

- Untuk detail API, lihat Memanggil di AWS SDK untuk Go Referensi API.

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   /**
    * Invokes a specific AWS Lambda function.
    *
    * @param awsLambda    an instance of {@link LambdaClient} to interact with
the AWS Lambda service
    * @param functionName the name of the AWS Lambda function to be invoked
    */
   public static void invokeFunction(LambdaClient awsLambda, String
functionName) {
       InvokeResponse res;
       try {
           // Need a SdkBytes instance for the payload.
           JSONObject jsonObj = new JSONObject();
           jsonObj.put("inputValue", "2000");
           String json = jsonObj.toString();
           SdkBytes payload = SdkBytes.fromUtf8String(json);

           InvokeRequest request = InvokeRequest.builder()
               .functionName(functionName)
               .payload(payload)
               .build();

           res = awsLambda.invoke(request);
           String value = res.payload().asUtf8String();
           System.out.println(value);

       } catch (LambdaException e) {
           System.err.println(e.getMessage());
           System.exit(1);
       }
   }
```

- Untuk detail API, lihat Memanggil di AWS SDK for Java 2.x Referensi API.

JavaScript

SDK untuk JavaScript (V3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
const invoke = async (funcName, payload) => {
  const client = new LambdaClient({});
  const command = new InvokeCommand({
    FunctionName: funcName,
    Payload: JSON.stringify(payload),
    LogType: LogType.Tail,
  });

  const { Payload, LogResult } = await client.send(command);
  const result = Buffer.from(Payload).toString();
  const logs = Buffer.from(LogResult, "base64").toString();
  return { logs, result };
};
```

- Untuk detail API, lihat Memanggil di AWS SDK untuk JavaScript Referensi API.

Kotlin

SDK untuk Kotlin

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
suspend fun invokeFunction(functionNameVal: String) {
    val json = """{"inputValue":"1000"}"""
    val byteArray = json.trimIndent().encodeToByteArray()
    val request =
        InvokeRequest {
            functionName = functionNameVal
            logType = LogType.Tail
            payload = byteArray
        }

    LambdaClient { region = "us-west-2" }.use { awsLambda ->
        val res = awsLambda.invoke(request)
        println("${res.payload?.toString(Charsets.UTF_8)}")
        println("The log result is ${res.logResult}")
    }
}
```

- Untuk detail API, lihat Memanggil di AWS SDK untuk referensi API Kotlin.

PHP

SDK untuk PHP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    public function invoke($functionName, $params, $logType = 'None')
    {
        return $this->lambdaClient->invoke([
            'FunctionName' => $functionName,
            'Payload' => json_encode($params),
            'LogType' => $logType,
        ]);
    }
```

- Untuk detail API, lihat Memanggil di AWS SDK untuk PHP Referensi API.

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
class LambdaWrapper:
    def __init__(self, lambda_client, iam_resource):
        self.lambda_client = lambda_client
        self.iam_resource = iam_resource


    def invoke_function(self, function_name, function_params, get_log=False):
        """
        Invokes a Lambda function.

        :param function_name: The name of the function to invoke.
        :param function_params: The parameters of the function as a dict. This
 dict
                                is serialized to JSON before it is sent to
 Lambda.
        :param get_log: When true, the last 4 KB of the execution log are
 included in
                        the response.
        :return: The response from the function invocation.
        """
        try:
            response = self.lambda_client.invoke(
                FunctionName=function_name,
                Payload=json.dumps(function_params),
                LogType="Tail" if get_log else "None",
            )
            logger.info("Invoked function %s.", function_name)
        except ClientError:
            logger.exception("Couldn't invoke function %s.", function_name)
            raise
        return response
```

- Untuk detail API, lihat Memanggil di AWS Referensi API SDK untuk Python (Boto3).

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
class LambdaWrapper
  attr_accessor :lambda_client, :cloudwatch_client, :iam_client

  def initialize
    @lambda_client = Aws::Lambda::Client.new
    @cloudwatch_client = Aws::CloudWatchLogs::Client.new(region: 'us-east-1')
    @iam_client = Aws::IAM::Client.new(region: 'us-east-1')
    @logger = Logger.new($stdout)
    @logger.level = Logger::WARN
  end

  # Invokes a Lambda function.
  # @param function_name [String] The name of the function to invoke.
  # @param payload [nil] Payload containing runtime parameters.
  # @return [Object] The response from the function invocation.
  def invoke_function(function_name, payload = nil)
    params = { function_name: function_name }
    params[:payload] = payload unless payload.nil?
    @lambda_client.invoke(params)
  rescue Aws::Lambda::Errors::ServiceException => e
    @logger.error("There was an error executing #{function_name}:\n
 #{e.message}")
  end
```

- Untuk detail API, lihat Memanggil di AWS SDK untuk Ruby Referensi API.

Rust

SDK for Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /** Invoke the lambda function using calculator InvokeArgs. */
    pub async fn invoke(&self, args: InvokeArgs) -> Result<InvokeOutput,
 anyhow::Error> {
        info!(?args, "Invoking {}", self.lambda_name);
        let payload = serde_json::to_string(&args)?;
        debug!(?payload, "Sending payload");
        self.lambda_client
            .invoke()
            .function_name(self.lambda_name.clone())
            .payload(Blob::new(payload))
            .send()
            .await
            .map_err(anyhow::Error::from)
    }

fn log_invoke_output(invoke: &InvokeOutput, message: &str) {
    if let Some(payload) = invoke.payload().cloned() {
        let payload = String::from_utf8(payload.into_inner());
        info!(?payload, message);
    } else {
        info!("Could not extract payload")
    }
    if let Some(logs) = invoke.log_result() {
        debug!(?logs, "Invoked function logs")
    } else {
        debug!("Invoked function had no logs")
    }
}
```

- Untuk detail API, lihat Memanggil di AWS Referensi SDK untuk Rust API.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   TRY.
       DATA(lv_json) = /aws1/cl_rt_util=>string_to_xstring(
         `{`  &&
           `"action": "increment",`  &&
           `"number": 10` &&
         `}` ).
       oo_result = lo_lmd->invoke(                 " oo_result is returned for
testing purposes. "
                 iv_functionname = iv_function_name
                 iv_payload = lv_json ).
       MESSAGE 'Lambda function invoked.' TYPE 'I'.
     CATCH /aws1/cx_lmdinvparamvalueex.
       MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
     CATCH /aws1/cx_lmdinvrequestcontex.
       MESSAGE 'Unable to parse request body as JSON.' TYPE 'E'.
     CATCH /aws1/cx_lmdinvalidzipfileex.
       MESSAGE 'The deployment package could not be unzipped.' TYPE 'E'.
     CATCH /aws1/cx_lmdrequesttoolargeex.
       MESSAGE 'Invoke request body JSON input limit was exceeded by the request
payload.' TYPE 'E'.
     CATCH /aws1/cx_lmdresourceconflictex.
       MESSAGE 'Resource already exists or another operation is in progress.'
TYPE 'E'.
     CATCH /aws1/cx_lmdresourcenotfoundex.
       MESSAGE 'The requested resource does not exist.' TYPE 'E'.
     CATCH /aws1/cx_lmdserviceexception.
       MESSAGE 'An internal problem was encountered by the AWS Lambda service.'
TYPE 'E'.
     CATCH /aws1/cx_lmdtoomanyrequestsex.
       MESSAGE 'The maximum request throughput was reached.' TYPE 'E'.
     CATCH /aws1/cx_lmdunsuppedmediatyp00.
```

```
        MESSAGE 'Invoke request body does not have JSON as its content type.'
 TYPE 'E'.
    ENDTRY.
```

- Untuk detail API, lihat Memanggil di AWS SDK untuk referensi API SAP ABAP.

Swift

SDK para Swift

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
import AWSClientRuntime
import AWSLambda
import Foundation

    /// Invoke the Lambda function to increment a value.
    ///
    /// - Parameters:
    ///   - lambdaClient: The `IAMClient` to use.
    ///   - number: The number to increment.
    ///
    /// - Throws: `ExampleError.noAnswerReceived`, `ExampleError.invokeError`
    ///
    /// - Returns: An integer number containing the incremented value.
    func invokeIncrement(lambdaClient: LambdaClient, number: Int) async throws ->
 Int {
        do {
            let incRequest = IncrementRequest(action: "increment", number:
 number)
            let incData = try! JSONEncoder().encode(incRequest)

            // Invoke the lambda function.

            let invokeOutput = try await lambdaClient.invoke(
                input: InvokeInput(
                    functionName: "lambda-basics-function",
```

```
                    payload: incData
                )
            )

            let response = try! JSONDecoder().decode(Response.self,
from:invokeOutput.payload!)

            guard let answer = response.answer else {
                throw ExampleError.noAnswerReceived
            }
            return answer

        } catch {
            throw ExampleError.invokeError
        }
    }
```

- Untuk detail API, lihat Memanggil di AWS SDK untuk referensi API Swift.

Untuk daftar lengkap AWS Panduan pengembang SDK dan contoh kode, lihat Menggunakan Lambda dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`ListFunctions`** dengan AWS SDK atau CLI

Contoh kode berikut menunjukkan cara menggunakan `ListFunctions`.

Contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Anda dapat melihat tindakan ini dalam konteks dalam contoh kode berikut:

- Pelajari dasar-dasarnya
- Membuat dasbor pemantauan dengan nama fungsi sebagai variabel

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
/// <summary>
/// Get a list of Lambda functions.
/// </summary>
/// <returns>A list of FunctionConfiguration objects.</returns>
public async Task<List<FunctionConfiguration>> ListFunctionsAsync()
{
    var functionList = new List<FunctionConfiguration>();

    var functionPaginator =
        _lambdaService.Paginators.ListFunctions(new ListFunctionsRequest());
    await foreach (var function in functionPaginator.Functions)
    {
        functionList.Add(function);
    }

    return functionList;
}
```

- Untuk detail API, lihat ListFunctions di AWS SDK untuk .NET Referensi API.

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
       Aws::Client::ClientConfiguration clientConfig;
       // Optional: Set to the AWS Region in which the bucket was created
(overrides config file).
       // clientConfig.region = "us-east-1";

   Aws::Lambda::LambdaClient client(clientConfig);

   std::vector<Aws::String> functions;
   Aws::String marker;

   do {
       Aws::Lambda::Model::ListFunctionsRequest request;
       if (!marker.empty()) {
           request.SetMarker(marker);
       }

       Aws::Lambda::Model::ListFunctionsOutcome outcome = client.ListFunctions(
               request);

       if (outcome.IsSuccess()) {
           const Aws::Lambda::Model::ListFunctionsResult &result =
outcome.GetResult();
           std::cout << result.GetFunctions().size()
                     << " lambda functions were retrieved." << std::endl;

           for (const Aws::Lambda::Model::FunctionConfiguration
&functionConfiguration: result.GetFunctions()) {
               functions.push_back(functionConfiguration.GetFunctionName());
               std::cout << functions.size() << "  "
                         << functionConfiguration.GetDescription() << std::endl;
               std::cout << "   "
                         <<
Aws::Lambda::Model::RuntimeMapper::GetNameForRuntime(
                                 functionConfiguration.GetRuntime()) << ": "
                         << functionConfiguration.GetHandler()
                         << std::endl;
           }
           marker = result.GetNextMarker();
       }
       else {
           std::cerr << "Error with Lambda::ListFunctions. "
                     << outcome.GetError().GetMessage()
                     << std::endl;
```

```
        }
    } while (!marker.empty());
```

- Untuk detail API, lihat ListFunctions di AWS SDK untuk C++ Referensi API.

CLI

AWS CLI

Untuk mengambil daftar fungsi Lambda

Berikut ini `list-functions` contoh menampilkan daftar semua fungsi untuk pengguna saat ini.

```
aws lambda list-functions
```

Output:

```
{
    "Functions": [
        {
            "TracingConfig": {
                "Mode": "PassThrough"
            },
            "Version": "$LATEST",
            "CodeSha256": "dBG9m8SGdmlEjw/JYXlhhvCrAv5TxvXsbL/RMr0fT/I=",
            "FunctionName": "helloworld",
            "MemorySize": 128,
            "RevisionId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
            "CodeSize": 294,
            "FunctionArn": "arn:aws:lambda:us-
west-2:123456789012:function:helloworld",
            "Handler": "helloworld.handler",
            "Role": "arn:aws:iam::123456789012:role/service-role/MyTestFunction-
role-zgur6bf4",
            "Timeout": 3,
            "LastModified": "2025-09-23T18:32:33.857+0000",
            "Runtime": "nodejs22.x",
            "Description": ""
        },
        {
```

```
            "TracingConfig": {
                "Mode": "PassThrough"
            },
            "Version": "$LATEST",
            "CodeSha256": "sU0cJ2/hOZevwV/lTxCuQqK3gDZP3i8gUoqUUVRmY6E=",
            "FunctionName": "my-function",
            "VpcConfig": {
                "SubnetIds": [],
                "VpcId": "",
                "SecurityGroupIds": []
            },
            "MemorySize": 256,
            "RevisionId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
            "CodeSize": 266,
            "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-
function",
            "Handler": "index.handler",
            "Role": "arn:aws:iam::123456789012:role/service-role/
helloWorldPython-role-uy3l9qyq",
            "Timeout": 3,
            "LastModified": "2025-10-01T16:47:28.490+0000",
            "Runtime": "nodejs22.x",
            "Description": ""
        },
        {
            "Layers": [
                {
                    "CodeSize": 41784542,
                    "Arn": "arn:aws:lambda:us-
west-2:420165488524:layer:AWSLambda-Python37-SciPy1x:2"
                },
                {
                    "CodeSize": 4121,
                    "Arn": "arn:aws:lambda:us-
west-2:123456789012:layer:pythonLayer:1"
                }
            ],
            "TracingConfig": {
                "Mode": "PassThrough"
            },
            "Version": "$LATEST",
            "CodeSha256": "ZQukCqxtkqFgyF2cU41Avj99TKQ/hNihPtDtRcc08mI=",
            "FunctionName": "my-python-function",
            "VpcConfig": {
```

```
                "SubnetIds": [],
                "VpcId": "",
                "SecurityGroupIds": []
            },
            "MemorySize": 128,
            "RevisionId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE33333",
            "CodeSize": 299,
            "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-
python-function",
            "Handler": "lambda_function.lambda_handler",
            "Role": "arn:aws:iam::123456789012:role/service-role/my-python-
function-role-z5g7dr6n",
            "Timeout": 3,
            "LastModified": "2025-10-01T19:40:41.643+0000",
            "Runtime": "python3.11",
            "Description": ""
        }
    ]
}
```

Untuk informasi lebih lanjut, lihat Konfigurasikan memori fungsi Lambda di AWS Panduan Pengembang Lambda.

- Untuk detail API, lihat ListFunctions di AWS CLI Referensi Perintah.

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
import (
 "bytes"
 "context"
 "encoding/json"
 "errors"
 "log"
```

```
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/lambda"
 "github.com/aws/aws-sdk-go-v2/service/lambda/types"
)

// FunctionWrapper encapsulates function actions used in the examples.
// It contains an AWS Lambda service client that is used to perform user actions.
type FunctionWrapper struct {
 LambdaClient *lambda.Client
}



// ListFunctions lists up to maxItems functions for the account. This function
 uses a
// lambda.ListFunctionsPaginator to paginate the results.
func (wrapper FunctionWrapper) ListFunctions(ctx context.Context, maxItems int)
 []types.FunctionConfiguration {
 var functions []types.FunctionConfiguration
 paginator := lambda.NewListFunctionsPaginator(wrapper.LambdaClient,
 &lambda.ListFunctionsInput{
  MaxItems: aws.Int32(int32(maxItems)),
 })
 for paginator.HasMorePages() && len(functions) < maxItems {
  pageOutput, err := paginator.NextPage(ctx)
  if err != nil {
   log.Panicf("Couldn't list functions for your account. Here's why: %v\n", err)
  }
  functions = append(functions, pageOutput.Functions...)
 }
 return functions
}
```

- Untuk detail API, lihat ListFunctions di AWS SDK untuk Go Referensi API.

JavaScript

SDK untuk JavaScript (V3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
const listFunctions = () => {
  const client = new LambdaClient({});
  const command = new ListFunctionsCommand({});

  return client.send(command);
};
```

- Untuk detail API, lihat ListFunctions di AWS SDK untuk JavaScript Referensi API.

PHP

SDK untuk PHP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    public function listFunctions($maxItems = 50, $marker = null)
    {
        if (is_null($marker)) {
            return $this->lambdaClient->listFunctions([
                'MaxItems' => $maxItems,
            ]);
        }

        return $this->lambdaClient->listFunctions([
            'Marker' => $marker,
```

```
            'MaxItems' => $maxItems,
        ]);
    }
```

- Untuk detail API, lihat ListFunctions di AWS SDK untuk PHP Referensi API.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Sampel ini menampilkan semua fungsi Lambda dengan ukuran kode yang diurutkan

```
Get-LMFunctionList | Sort-Object -Property CodeSize | Select-Object FunctionName,
 RunTime, Timeout, CodeSize
```

Output:

```
FunctionName                                                  Runtime   Timeout
 CodeSize
------------                                                  -------   -------
 --------
test                                                          python2.7       3
   243
MylambdaFunction123                                           python3.8     600
   659
myfuncpython1                                                 python3.8     303
   675
```

- Untuk detail API, lihat ListFunctions di Alat AWS untuk PowerShell Referensi Cmdlet (V4).

Alat untuk PowerShell V5

Contoh 1: Sampel ini menampilkan semua fungsi Lambda dengan ukuran kode yang diurutkan

```
Get-LMFunctionList | Sort-Object -Property CodeSize | Select-Object FunctionName,
 RunTime, Timeout, CodeSize
```

Output:

```
FunctionName                                                  Runtime   Timeout
 CodeSize
```

```
------------                                                    -------   -------
 --------
test                                                            python2.7       3
   243
MylambdaFunction123                                             python3.8     600
   659
myfuncpython1                                                   python3.8     303
   675
```

- Untuk detail API, lihat ListFunctions di Alat AWS untuk PowerShell Referensi Cmdlet (V5).

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
class LambdaWrapper:
    def __init__(self, lambda_client, iam_resource):
        self.lambda_client = lambda_client
        self.iam_resource = iam_resource


    def list_functions(self):
        """
        Lists the Lambda functions for the current account.
        """
        try:
            func_paginator = self.lambda_client.get_paginator("list_functions")
            for func_page in func_paginator.paginate():
                for func in func_page["Functions"]:
                    print(func["FunctionName"])
                    desc = func.get("Description")
                    if desc:
                        print(f"\t{desc}")
                    print(f"\t{func['Runtime']}: {func['Handler']}")
        except ClientError as err:
            logger.error(
```

```
                "Couldn't list functions. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- Untuk detail API, lihat ListFunctions di AWS Referensi API SDK untuk Python (Boto3).

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
class LambdaWrapper
  attr_accessor :lambda_client, :cloudwatch_client, :iam_client

  def initialize
    @lambda_client = Aws::Lambda::Client.new
    @cloudwatch_client = Aws::CloudWatchLogs::Client.new(region: 'us-east-1')
    @iam_client = Aws::IAM::Client.new(region: 'us-east-1')
    @logger = Logger.new($stdout)
    @logger.level = Logger::WARN
  end

  # Lists the Lambda functions for the current account.
  def list_functions
    functions = []
    @lambda_client.list_functions.each do |response|
      response['functions'].each do |function|
        functions.append(function['function_name'])
      end
    end
    functions
  rescue Aws::Lambda::Errors::ServiceException => e
    @logger.error("There was an error listing functions:\n #{e.message}")
```

```
  end
```

- Untuk detail API, lihat ListFunctions di AWS SDK untuk Ruby Referensi API.

Rust

SDK for Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   /** List all Lambda functions in the current Region. */
   pub async fn list_functions(&self) -> Result<ListFunctionsOutput,
anyhow::Error> {
       info!("Listing lambda functions");
       self.lambda_client
           .list_functions()
           .send()
           .await
           .map_err(anyhow::Error::from)
   }
```

- Untuk detail API, lihat ListFunctions di AWS Referensi SDK untuk Rust API.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    TRY.
        oo_result = lo_lmd->listfunctions( ).       " oo_result is returned for
testing purposes. "
        DATA(lt_functions) = oo_result->get_functions( ).
        MESSAGE 'Retrieved list of Lambda functions.' TYPE 'I'.
      CATCH /aws1/cx_lmdinvparamvalueex.
        MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
      CATCH /aws1/cx_lmdserviceexception.
        MESSAGE 'An internal problem was encountered by the AWS Lambda service.'
TYPE 'E'.
      CATCH /aws1/cx_lmdtoomanyrequestsex.
        MESSAGE 'The maximum request throughput was reached.' TYPE 'E'.
    ENDTRY.
```

- Untuk detail API, lihat ListFunctions di AWS SDK untuk referensi API SAP ABAP.

Swift

SDK para Swift

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
import AWSClientRuntime
import AWSLambda
import Foundation

    /// Returns an array containing the names of all AWS Lambda functions
    /// available to the user.
    ///
    /// - Parameter lambdaClient: The `IAMClient` to use.
    ///
    /// - Throws: `ExampleError.listFunctionsError`
    ///
    /// - Returns: An array of lambda function name strings.
    func getFunctionNames(lambdaClient: LambdaClient) async throws -> [String] {
        let pages = lambdaClient.listFunctionsPaginated(
```

```
            input: ListFunctionsInput()
        )

        var functionNames: [String] = []

        for try await page in pages {
            guard let functions = page.functions else {
                throw ExampleError.listFunctionsError
            }

            for function in functions {
                functionNames.append(function.functionName ?? "<unknown>")
            }
        }

        return functionNames
    }
```

- Untuk detail API, lihat ListFunctions di AWS SDK untuk referensi API Swift.

Untuk daftar lengkap AWS Panduan pengembang SDK dan contoh kode, lihat Menggunakan Lambda dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`ListProvisionedConcurrencyConfigs`** dengan CLI

Contoh kode berikut menunjukkan cara menggunakan `ListProvisionedConcurrencyConfigs`.

CLI

AWS CLI

Untuk mendapatkan daftar konfigurasi konkurensi yang disediakan

Berikut ini `list-provisioned-concurrency-configs` contoh mencantumkan konfigurasi konkurensi yang disediakan untuk fungsi yang ditentukan.

```
aws lambda list-provisioned-concurrency-configs \
    --function-name my-function
```

Output:

```
{
    "ProvisionedConcurrencyConfigs": [
        {
            "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-
function:GREEN",
            "RequestedProvisionedConcurrentExecutions": 100,
            "AvailableProvisionedConcurrentExecutions": 100,
            "AllocatedProvisionedConcurrentExecutions": 100,
            "Status": "READY",
            "LastModified": "2019-12-31T20:29:00+0000"
        },
        {
            "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-
function:BLUE",
            "RequestedProvisionedConcurrentExecutions": 100,
            "AvailableProvisionedConcurrentExecutions": 100,
            "AllocatedProvisionedConcurrentExecutions": 100,
            "Status": "READY",
            "LastModified": "2019-12-31T20:28:49+0000"
        }
    ]
}
```

Untuk informasi lebih lanjut, lihat Mengkonfigurasi konkurensi yang disediakan untuk Fungsi Peristiwa di AWS Panduan Pengembang Lambda.

- Untuk detail API, lihat ListProvisionedConcurrencyConfigs di AWS CLI Referensi Perintah.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Contoh ini mengambil daftar konfigurasi konkurensi yang disediakan untuk fungsi Lambda.

```
Get-LMProvisionedConcurrencyConfigList -FunctionName "MylambdaFunction123"
```

- Untuk detail API, lihat ListProvisionedConcurrencyConfigs di Alat AWS untuk PowerShell Referensi Cmdlet (V4).

Alat untuk PowerShell V5

Contoh 1: Contoh ini mengambil daftar konfigurasi konkurensi yang disediakan untuk fungsi Lambda.

```
Get-LMProvisionedConcurrencyConfigList -FunctionName "MylambdaFunction123"
```

- Untuk detail API, lihat ListProvisionedConcurrencyConfigs di Alat AWS untuk PowerShell Referensi Cmdlet (V5).

Untuk daftar lengkap AWS Panduan pengembang SDK dan contoh kode, lihat Menggunakan Lambda dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`ListTags`** dengan CLI

Contoh kode berikut menunjukkan cara menggunakan `ListTags`.

CLI

AWS CLI

Untuk mengambil daftar tag untuk fungsi Lambda

Berikut ini `list-tags` contoh menampilkan tag yang dilampirkan ke `my-function` Fungsi lambda.

```
aws lambda list-tags \
    --resource arn:aws:lambda:us-west-2:123456789012:function:my-function
```

Output:

```
{
    "Tags": {
        "Category": "Web Tools",
        "Department": "Sales"
    }
}
```

Untuk informasi lebih lanjut, lihat Menandai Fungsi Lambda di AWS Panduan Pengembang Lambda.

- Untuk detail API, lihat ListTags di AWS CLI Referensi Perintah.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Mengambil tag dan nilainya yang saat ini ditetapkan pada fungsi yang ditentukan.

```
Get-LMResourceTag -Resource "arn:aws:lambda:us-
west-2:123456789012:function:MyFunction"
```

Output:

```
Key        Value
---        -----
California Sacramento
Oregon     Salem
Washington Olympia
```

- Untuk detail API, lihat ListTags di Alat AWS untuk PowerShell Referensi Cmdlet (V4).

Alat untuk PowerShell V5

Contoh 1: Mengambil tag dan nilainya yang saat ini ditetapkan pada fungsi yang ditentukan.

```
Get-LMResourceTag -Resource "arn:aws:lambda:us-
west-2:123456789012:function:MyFunction"
```

Output:

```
Key        Value
---        -----
California Sacramento
Oregon     Salem
Washington Olympia
```

- Untuk detail API, lihat ListTags di Alat AWS untuk PowerShell Referensi Cmdlet (V5).

Untuk daftar lengkap AWS Panduan pengembang SDK dan contoh kode, lihat Menggunakan Lambda dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`ListVersionsByFunction`** dengan CLI

Contoh kode berikut menunjukkan cara menggunakan `ListVersionsByFunction`.

CLI

AWS CLI

Untuk mengambil daftar versi fungsi

Berikut ini `list-versions-by-function` contoh menampilkan daftar versi untuk `my-function` Fungsi lambda.

```
aws lambda list-versions-by-function \
    --function-name my-function
```

Output:

```
{
    "Versions": [
        {
            "TracingConfig": {
                "Mode": "PassThrough"
            },
            "Version": "$LATEST",
            "CodeSha256": "sU0cJ2/hOZevwV/lTxCuQqK3gDZP3i8gUoqUUVRmY6E=",
            "FunctionName": "my-function",
            "VpcConfig": {
                "SubnetIds": [],
                "VpcId": "",
                "SecurityGroupIds": []
            },
            "MemorySize": 256,
            "RevisionId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
            "CodeSize": 266,
            "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-
function:$LATEST",
            "Handler": "index.handler",
            "Role": "arn:aws:iam::123456789012:role/service-role/
helloWorldPython-role-uy3l9qyq",
            "Timeout": 3,
            "LastModified": "2019-10-01T16:47:28.490+0000",
            "Runtime": "nodejs10.x",
```

```
            "Description": ""
        },
        {
            "TracingConfig": {
                "Mode": "PassThrough"
            },
            "Version": "1",
            "CodeSha256": "5tT2qgzYUHoqwR616pZ2dpkn/0J1FrzJmlKidWaaCgk=",
            "FunctionName": "my-function",
            "VpcConfig": {
                "SubnetIds": [],
                "VpcId": "",
                "SecurityGroupIds": []
            },
            "MemorySize": 256,
            "RevisionId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
            "CodeSize": 304,
            "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-
function:1",
            "Handler": "index.handler",
            "Role": "arn:aws:iam::123456789012:role/service-role/
helloWorldPython-role-uy3l9qyq",
            "Timeout": 3,
            "LastModified": "2019-09-26T20:28:40.438+0000",
            "Runtime": "nodejs10.x",
            "Description": "new version"
        },
        {
            "TracingConfig": {
                "Mode": "PassThrough"
            },
            "Version": "2",
            "CodeSha256": "sU0cJ2/hOZevwV/lTxCuQqK3gDZP3i8gUoqUUVRmY6E=",
            "FunctionName": "my-function",
            "VpcConfig": {
                "SubnetIds": [],
                "VpcId": "",
                "SecurityGroupIds": []
            },
            "MemorySize": 256,
            "RevisionId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE33333",
            "CodeSize": 266,
            "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-
function:2",
```

```
            "Handler": "index.handler",
            "Role": "arn:aws:iam::123456789012:role/service-role/
helloWorldPython-role-uy3l9qyq",
            "Timeout": 3,
            "LastModified": "2019-10-01T16:47:28.490+0000",
            "Runtime": "nodejs10.x",
            "Description": "newer version"
        }
    ]
}
```

Untuk informasi lebih lanjut, lihat Mengkonfigurasi AWS Alias Fungsi Lambda di AWS Panduan Pengembang Lambda.

- Untuk detail API, lihat ListVersionsByFunction di AWS CLI Referensi Perintah.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Contoh ini mengembalikan daftar konfigurasi khusus versi untuk setiap versi Fungsi Lambda.

```
Get-LMVersionsByFunction -FunctionName "MylambdaFunction123"
```

Output:

```
FunctionName        Runtime   MemorySize Timeout CodeSize LastModified
      RoleName
------------        -------   ---------- ------- -------- ------------
      --------
MylambdaFunction123 python3.8        128     600      659
 2020-01-10T03:20:56.390+0000 lambda
MylambdaFunction123 python3.8        128       5     1426
 2019-12-25T09:19:02.238+0000 lambda
MylambdaFunction123 python3.8        128       5     1426
 2019-12-25T09:39:36.779+0000 lambda
MylambdaFunction123 python3.8        128     600     1426
 2019-12-25T09:52:59.872+0000 lambda
```

- Untuk detail API, lihat ListVersionsByFunction di Alat AWS untuk PowerShell Referensi Cmdlet (V4).

Alat untuk PowerShell V5

Contoh 1: Contoh ini mengembalikan daftar konfigurasi khusus versi untuk setiap versi Fungsi Lambda.

```
Get-LMVersionsByFunction -FunctionName "MylambdaFunction123"
```

Output:

```
FunctionName        Runtime   MemorySize Timeout CodeSize LastModified
      RoleName
------------        -------   ---------- ------- -------- ------------
      --------
MylambdaFunction123 python3.8        128     600      659
 2020-01-10T03:20:56.390+0000 lambda
MylambdaFunction123 python3.8        128       5     1426
 2019-12-25T09:19:02.238+0000 lambda
MylambdaFunction123 python3.8        128       5     1426
 2019-12-25T09:39:36.779+0000 lambda
MylambdaFunction123 python3.8        128     600     1426
 2019-12-25T09:52:59.872+0000 lambda
```

- Untuk detail API, lihat ListVersionsByFunction di Alat AWS untuk PowerShell Referensi Cmdlet (V5).

Untuk daftar lengkap AWS Panduan pengembang SDK dan contoh kode, lihat Menggunakan Lambda dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`PublishVersion`** dengan CLI

Contoh kode berikut menunjukkan cara menggunakan `PublishVersion`.

CLI

AWS CLI

Untuk menerbitkan versi baru dari suatu fungsi

Berikut ini `publish-version` contoh menerbitkan versi baru dari `my-function` Fungsi lambda.

```
aws lambda publish-version \
    --function-name my-function
```

Output:

```
{
    "TracingConfig": {
        "Mode": "PassThrough"
    },
    "CodeSha256": "dBG9m8SGdmlEjw/JYXlhhvCrAv5TxvXsbL/RMr0fT/I=",
    "FunctionName": "my-function",
    "CodeSize": 294,
    "RevisionId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
    "MemorySize": 128,
    "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-
function:3",
    "Version": "2",
    "Role": "arn:aws:iam::123456789012:role/service-role/MyTestFunction-role-
zgur6bf4",
    "Timeout": 3,
    "LastModified": "2019-09-23T18:32:33.857+0000",
    "Handler": "my-function.handler",
    "Runtime": "nodejs10.x",
    "Description": ""
}
```

Untuk informasi lebih lanjut, lihat Mengkonfigurasi AWS Alias Fungsi Lambda di AWS Panduan Pengembang Lambda.

- Untuk detail API, lihat PublishVersion di AWS CLI Referensi Perintah.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Contoh ini membuat versi untuk snapshot yang ada dari Kode Fungsi Lambda

```
Publish-LMVersion -FunctionName "MylambdaFunction123" -Description "Publishing
 Existing Snapshot of function code as a  new version through Powershell"
```

- Untuk detail API, lihat PublishVersion di Alat AWS untuk PowerShell Referensi Cmdlet (V4).

Alat untuk PowerShell V5

Contoh 1: Contoh ini membuat versi untuk snapshot yang ada dari Kode Fungsi Lambda

```
Publish-LMVersion -FunctionName "MylambdaFunction123" -Description "Publishing
 Existing Snapshot of function code as a  new version through Powershell"
```

- Untuk detail API, lihat PublishVersion di Alat AWS untuk PowerShell Referensi Cmdlet (V5).

Untuk daftar lengkap AWS Panduan pengembang SDK dan contoh kode, lihat Menggunakan Lambda dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`PutFunctionConcurrency`** dengan CLI

Contoh kode berikut menunjukkan cara menggunakan `PutFunctionConcurrency`.

CLI

AWS CLI

Untuk mengonfigurasi batas konkurensi yang dicadangkan untuk suatu fungsi

Berikut ini `put-function-concurrency` contoh mengonfigurasi 100 eksekusi bersamaan yang dicadangkan untuk `my-function` fungsi.

```
aws lambda put-function-concurrency \
    --function-name  my-function  \
    --reserved-concurrent-executions 100
```

Output:

```
{
    "ReservedConcurrentExecutions": 100
}
```

Untuk informasi lebih lanjut, lihat Memesan Konkurensi untuk Fungsi Lambda di AWS Panduan Pengembang Lambda.

- Untuk detail API, lihat PutFunctionConcurrency di AWS CLI Referensi Perintah.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Contoh ini menerapkan pengaturan konkurensi untuk Fungsi secara keseluruhan.

```
Write-LMFunctionConcurrency -FunctionName "MylambdaFunction123" -
ReservedConcurrentExecution 100
```

- Untuk detail API, lihat PutFunctionConcurrency di Alat AWS untuk PowerShell Referensi Cmdlet (V4).

Alat untuk PowerShell V5

Contoh 1: Contoh ini menerapkan pengaturan konkurensi untuk Fungsi secara keseluruhan.

```
Write-LMFunctionConcurrency -FunctionName "MylambdaFunction123" -
ReservedConcurrentExecution 100
```

- Untuk detail API, lihat PutFunctionConcurrency di Alat AWS untuk PowerShell Referensi Cmdlet (V5).

Untuk daftar lengkap AWS Panduan pengembang SDK dan contoh kode, lihat Menggunakan Lambda dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`PutProvisionedConcurrencyConfig`** dengan CLI

Contoh kode berikut menunjukkan cara menggunakan `PutProvisionedConcurrencyConfig`.

CLI

AWS CLI

Untuk mengalokasikan konkurensi yang disediakan

Berikut ini `put-provisioned-concurrency-config` contoh mengalokasikan 100 konkurensi yang disediakan untuk BLUE alias dari fungsi yang ditentukan.

```
aws lambda put-provisioned-concurrency-config \
    --function-name my-function \
    --qualifier BLUE \
```

```
    --provisioned-concurrent-executions 100
```

Output:

```
{
    "RequestedProvisionedConcurrentExecutions": 100,
    "AllocatedProvisionedConcurrentExecutions": 0,
    "Status": "IN_PROGRESS",
    "LastModified": "2019-11-21T19:32:12+0000"
}
```

Untuk informasi lebih lanjut, lihat Mengkonfigurasi konkurensi yang disediakan untuk Fungsi Peristiwa di AWS Panduan Pengembang Lambda.

- Untuk detail API, lihat PutProvisionedConcurrencyConfig di AWS CLI Referensi Perintah.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Contoh ini menambahkan konfigurasi konkurensi yang disediakan ke Alias Fungsi

```
Write-LMProvisionedConcurrencyConfig -FunctionName "MylambdaFunction123" -
ProvisionedConcurrentExecution 20 -Qualifier "NewAlias1"
```

- Untuk detail API, lihat PutProvisionedConcurrencyConfig di Alat AWS untuk PowerShell Referensi Cmdlet (V4).

Alat untuk PowerShell V5

Contoh 1: Contoh ini menambahkan konfigurasi konkurensi yang disediakan ke Alias Fungsi

```
Write-LMProvisionedConcurrencyConfig -FunctionName "MylambdaFunction123" -
ProvisionedConcurrentExecution 20 -Qualifier "NewAlias1"
```

- Untuk detail API, lihat PutProvisionedConcurrencyConfig di Alat AWS untuk PowerShell Referensi Cmdlet (V5).

Untuk daftar lengkap AWS Panduan pengembang SDK dan contoh kode, lihat Menggunakan Lambda dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`RemovePermission`** dengan CLI

Contoh kode berikut menunjukkan cara menggunakan `RemovePermission`.

CLI

AWS CLI

Untuk menghapus izin dari fungsi Lambda yang ada

Berikut ini `remove-permission` contoh menghapus izin untuk memanggil fungsi bernama `my-function`.

```
aws lambda remove-permission \
    --function-name my-function \
    --statement-id sns
```

Perintah ini tidak menghasilkan output.

Untuk informasi lebih lanjut, lihat Menggunakan Resource-based Kebijakan untuk AWS Lambda di AWS Panduan Pengembang Lambda.

- Untuk detail API, lihat RemovePermission di AWS CLI Referensi Perintah.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Contoh ini menghapus kebijakan fungsi untuk yang ditentukan StatementId dari Fungsi Lambda.

```
$policy =  Get-LMPolicy -FunctionName "MylambdaFunction123" -Select Policy |
 ConvertFrom-Json| Select-Object -ExpandProperty Statement
Remove-LMPermission -FunctionName "MylambdaFunction123" -StatementId
 $policy[0].Sid
```

- Untuk detail API, lihat RemovePermission di Alat AWS untuk PowerShell Referensi Cmdlet (V4).

Alat untuk PowerShell V5

Contoh 1: Contoh ini menghapus kebijakan fungsi untuk yang ditentukan StatementId dari Fungsi Lambda.

```
$policy =  Get-LMPolicy -FunctionName "MylambdaFunction123" -Select Policy |
 ConvertFrom-Json| Select-Object -ExpandProperty Statement
Remove-LMPermission -FunctionName "MylambdaFunction123" -StatementId
 $policy[0].Sid
```

- Untuk detail API, lihat RemovePermission di Alat AWS untuk PowerShell Referensi Cmdlet (V5).

Untuk daftar lengkap AWS Panduan pengembang SDK dan contoh kode, lihat Menggunakan Lambda dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`TagResource`** dengan CLI

Contoh kode berikut menunjukkan cara menggunakan `TagResource`.

CLI

AWS CLI

Untuk menambahkan tag ke fungsi Lambda yang ada

Berikut ini `tag-resource` contoh menambahkan tag dengan nama kunci `DEPARTMENT` dan nilai `Department A` ke fungsi Lambda yang ditentukan.

```
aws lambda tag-resource \
    --resource arn:aws:lambda:us-west-2:123456789012:function:my-function \
    --tags "DEPARTMENT=Department A"
```

Perintah ini tidak menghasilkan output.

Untuk informasi lebih lanjut, lihat Menandai Fungsi Lambda di AWS Panduan Pengembang Lambda.

- Untuk detail API, lihat TagResource di AWS CLI Referensi Perintah.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Menambahkan tiga tag (Washington, Oregon dan California) dan nilai terkait mereka ke fungsi tertentu yang diidentifikasi oleh ARN-nya.

```
Add-LMResourceTag -Resource "arn:aws:lambda:us-
west-2:123456789012:function:MyFunction" -Tag @{ "Washington" = "Olympia";
 "Oregon" = "Salem"; "California" = "Sacramento" }
```

- Untuk detail API, lihat TagResource di Alat AWS untuk PowerShell Referensi Cmdlet (V4).

Alat untuk PowerShell V5

Contoh 1: Menambahkan tiga tag (Washington, Oregon dan California) dan nilai terkait mereka ke fungsi tertentu yang diidentifikasi oleh ARN-nya.

```
Add-LMResourceTag -Resource "arn:aws:lambda:us-
west-2:123456789012:function:MyFunction" -Tag @{ "Washington" = "Olympia";
 "Oregon" = "Salem"; "California" = "Sacramento" }
```

- Untuk detail API, lihat TagResource di Alat AWS untuk PowerShell Referensi Cmdlet (V5).

Untuk daftar lengkap AWS Panduan pengembang SDK dan contoh kode, lihat Menggunakan Lambda dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`UntagResource`** dengan CLI

Contoh kode berikut menunjukkan cara menggunakan `UntagResource`.

CLI

AWS CLI

Untuk menghapus tag dari fungsi Lambda yang ada

Berikut ini `untag-resource` contoh menghapus tag dengan nama kunci `DEPARTMENT` tag dari `my-function` Fungsi lambda.

```
aws lambda untag-resource \
```

```
    --resource arn:aws:lambda:us-west-2:123456789012:function:my-function \
    --tag-keys DEPARTMENT
```

Perintah ini tidak menghasilkan output.

Untuk informasi lebih lanjut, lihat Menandai Fungsi Lambda di AWS Panduan Pengembang Lambda.

- Untuk detail API, lihat UntagResource di AWS CLI Referensi Perintah.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Menghapus tag yang disediakan dari fungsi. Cmdlet akan meminta konfirmasi sebelum melanjutkan kecuali sakelar -Force ditentukan. Satu panggilan dilakukan ke layanan untuk menghapus tag.

```
Remove-LMResourceTag -Resource "arn:aws:lambda:us-
west-2:123456789012:function:MyFunction" -TagKey
 "Washington","Oregon","California"
```

Contoh 2: Menghapus tag yang disediakan dari fungsi. Cmdlet akan meminta konfirmasi sebelum melanjutkan kecuali sakelar -Force ditentukan. Setelah panggilan ke layanan dilakukan per tag yang disediakan.

```
"Washington","Oregon","California" | Remove-LMResourceTag -Resource
 "arn:aws:lambda:us-west-2:123456789012:function:MyFunction"
```

- Untuk detail API, lihat UntagResource di Alat AWS untuk PowerShell Referensi Cmdlet (V4).

Alat untuk PowerShell V5

Contoh 1: Menghapus tag yang disediakan dari fungsi. Cmdlet akan meminta konfirmasi sebelum melanjutkan kecuali sakelar -Force ditentukan. Satu panggilan dilakukan ke layanan untuk menghapus tag.

```
Remove-LMResourceTag -Resource "arn:aws:lambda:us-
west-2:123456789012:function:MyFunction" -TagKey
 "Washington","Oregon","California"
```

Contoh 2: Menghapus tag yang disediakan dari fungsi. Cmdlet akan meminta konfirmasi sebelum melanjutkan kecuali sakelar -Force ditentukan. Setelah panggilan ke layanan dilakukan per tag yang disediakan.

```
"Washington","Oregon","California" | Remove-LMResourceTag -Resource
 "arn:aws:lambda:us-west-2:123456789012:function:MyFunction"
```

- Untuk detail API, lihat UntagResource di Alat AWS untuk PowerShell Referensi Cmdlet (V5).

Untuk daftar lengkap AWS Panduan pengembang SDK dan contoh kode, lihat Menggunakan Lambda dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`UpdateAlias`** dengan CLI

Contoh kode berikut menunjukkan cara menggunakan `UpdateAlias`.

CLI

AWS CLI

Untuk memperbarui alias fungsi

Berikut ini `update-alias` contoh memperbarui alias bernama `LIVE` untuk menunjuk ke versi 3 dari `my-function` Fungsi lambda.

```
aws lambda update-alias \
    --function-name my-function \
    --function-version 3 \
    --name LIVE
```

Output:

```
{
    "FunctionVersion": "3",
    "Name": "LIVE",
    "AliasArn": "arn:aws:lambda:us-west-2:123456789012:function:my-
function:LIVE",
    "RevisionId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
    "Description": "alias for live version of function"
```

```
}
```

Untuk informasi lebih lanjut, lihat Mengkonfigurasi AWS Alias Fungsi Lambda di AWS Panduan Pengembang Lambda.

- Untuk detail API, lihat UpdateAlias di AWS CLI Referensi Perintah.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Contoh ini memperbarui Konfigurasi Alias fungsi Lambda yang ada. Ini memperbarui RoutingConfiguration nilai untuk menggeser 60% (0,6) lalu lintas ke versi 1

```
Update-LMAlias -FunctionName "MylambdaFunction123" -Description
 " Alias for version 2" -FunctionVersion 2 -Name "newlabel1" -
RoutingConfig_AdditionalVersionWeight @{Name="1";Value="0.6}
```

- Untuk detail API, lihat UpdateAlias di Alat AWS untuk PowerShell Referensi Cmdlet (V4).

Alat untuk PowerShell V5

Contoh 1: Contoh ini memperbarui Konfigurasi Alias fungsi Lambda yang ada. Ini memperbarui RoutingConfiguration nilai untuk menggeser 60% (0,6) lalu lintas ke versi 1

```
Update-LMAlias -FunctionName "MylambdaFunction123" -Description
 " Alias for version 2" -FunctionVersion 2 -Name "newlabel1" -
RoutingConfig_AdditionalVersionWeight @{Name="1";Value="0.6}
```

- Untuk detail API, lihat UpdateAlias di Alat AWS untuk PowerShell Referensi Cmdlet (V5).

Untuk daftar lengkap AWS Panduan pengembang SDK dan contoh kode, lihat Menggunakan Lambda dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`UpdateFunctionCode`** dengan AWS SDK atau CLI

Contoh kode berikut menunjukkan cara menggunakan `UpdateFunctionCode`.

Contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Anda dapat melihat tindakan ini dalam konteks dalam contoh kode berikut:

- Pelajari dasar-dasarnya

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Update an existing Lambda function.
    /// </summary>
    /// <param name="functionName">The name of the Lambda function to update.</
param>
    /// <param name="bucketName">The bucket where the zip file containing
    /// the Lambda function code is stored.</param>
    /// <param name="key">The key name of the source code file.</param>
    /// <returns>Async Task.</returns>
    public async Task UpdateFunctionCodeAsync(
        string functionName,
        string bucketName,
        string key)
    {
        var functionCodeRequest = new UpdateFunctionCodeRequest
        {
            FunctionName = functionName,
            Publish = true,
            S3Bucket = bucketName,
            S3Key = key,
        };

        var response = await
 _lambdaService.UpdateFunctionCodeAsync(functionCodeRequest);
       Console.WriteLine($"The Function was last modified at
 {response.LastModified}.");
    }
```

- Untuk detail API, lihat UpdateFunctionCode di AWS SDK untuk .NET Referensi API.

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region in which the bucket was created
 (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::Lambda::LambdaClient client(clientConfig);

        Aws::Lambda::Model::UpdateFunctionCodeRequest request;
        request.SetFunctionName(LAMBDA_NAME);
        std::ifstream ifstream(CALCULATOR_LAMBDA_CODE.c_str(),
                               std::ios_base::in | std::ios_base::binary);
        if (!ifstream.is_open()) {
            std::cerr << "Error opening file " << INCREMENT_LAMBDA_CODE << "." <<
 std::endl;

#if USE_CPP_LAMBDA_FUNCTION
            std::cerr
                    << "The cpp Lambda function must be built following the
 instructions in the cpp_lambda/README.md file. "
                    << std::endl;
#endif
            deleteLambdaFunction(client);
            deleteIamRole(clientConfig);
            return false;
        }

        Aws::StringStream buffer;
        buffer << ifstream.rdbuf();
        request.SetZipFile(
                Aws::Utils::ByteBuffer((unsigned char *) buffer.str().c_str(),
```

```
                                        buffer.str().length()));
        request.SetPublish(true);

        Aws::Lambda::Model::UpdateFunctionCodeOutcome outcome =
client.UpdateFunctionCode(
                request);

        if (outcome.IsSuccess()) {
            std::cout << "The lambda code was successfully updated." <<
std::endl;
        }
        else {
            std::cerr << "Error with Lambda::UpdateFunctionCode. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
        }
```

- Untuk detail API, lihat UpdateFunctionCode di AWS SDK untuk C++ Referensi API.

CLI

AWS CLI

Untuk memperbarui kode fungsi Lambda

Berikut ini `update-function-code` contoh menggantikan kode versi yang tidak dipublikasikan ($LATEST) dari `my-function` berfungsi dengan isi file zip yang ditentukan.

```
aws lambda update-function-code \
    --function-name  my-function \
    --zip-file fileb://my-function.zip
```

Output:

```
{
    "FunctionName": "my-function",
    "LastModified": "2019-09-26T20:28:40.438+0000",
    "RevisionId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
    "MemorySize": 256,
    "Version": "$LATEST",
```

```
    "Role": "arn:aws:iam::123456789012:role/service-role/my-function-role-
uy3l9qyq",
    "Timeout": 3,
    "Runtime": "nodejs10.x",
    "TracingConfig": {
        "Mode": "PassThrough"
    },
    "CodeSha256": "5tT2qgzYUHaqwR716pZ2dpkn/0J1FrzJmlKidWoaCgk=",
    "Description": "",
    "VpcConfig": {
        "SubnetIds": [],
        "VpcId": "",
        "SecurityGroupIds": []
    },
    "CodeSize": 304,
    "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-function",
    "Handler": "index.handler"
}
```

Untuk informasi lebih lanjut, lihat AWS Konfigurasi Fungsi Lambda di AWS Panduan Pengembang Lambda.

- Untuk detail API, lihat UpdateFunctionCode di AWS CLI Referensi Perintah.

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
import (
 "bytes"
 "context"
 "encoding/json"
 "errors"
 "log"
 "time"
```

```
 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/lambda"
 "github.com/aws/aws-sdk-go-v2/service/lambda/types"
)

// FunctionWrapper encapsulates function actions used in the examples.
// It contains an AWS Lambda service client that is used to perform user actions.
type FunctionWrapper struct {
 LambdaClient *lambda.Client
}



// UpdateFunctionCode updates the code for the Lambda function specified by
 functionName.
// The existing code for the Lambda function is entirely replaced by the code in
 the
// zipPackage buffer. After the update action is called, a
 lambda.FunctionUpdatedV2Waiter
// is used to wait until the update is successful.
func (wrapper FunctionWrapper) UpdateFunctionCode(ctx context.Context,
 functionName string, zipPackage *bytes.Buffer) types.State {
 var state types.State
 _, err := wrapper.LambdaClient.UpdateFunctionCode(ctx,
 &lambda.UpdateFunctionCodeInput{
  FunctionName: aws.String(functionName), ZipFile: zipPackage.Bytes(),
 })
 if err != nil {
  log.Panicf("Couldn't update code for function %v. Here's why: %v\n",
 functionName, err)
 } else {
  waiter := lambda.NewFunctionUpdatedV2Waiter(wrapper.LambdaClient)
  funcOutput, err := waiter.WaitForOutput(ctx, &lambda.GetFunctionInput{
   FunctionName: aws.String(functionName)}, 1*time.Minute)
  if err != nil {
   log.Panicf("Couldn't wait for function %v to be active. Here's why: %v\n",
 functionName, err)
  } else {
   state = funcOutput.Configuration.State
  }
 }
 return state
}
```

- Untuk detail API, lihat UpdateFunctionCode di AWS SDK untuk Go Referensi API.

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   /**
    * Updates the code for an AWS Lambda function.
    *
    * @param awsLambda  the AWS Lambda client
    * @param functionName the name of the Lambda function to update
    * @param bucketName the name of the S3 bucket where the function code is
located
    * @param key the key (file name) of the function code in the S3 bucket
    * @throws LambdaException if there is an error updating the function code
    */
   public static void updateFunctionCode(LambdaClient awsLambda, String
functionName, String bucketName, String key) {
       try {
           LambdaWaiter waiter = awsLambda.waiter();
           UpdateFunctionCodeRequest functionCodeRequest =
UpdateFunctionCodeRequest.builder()
                .functionName(functionName)
                .publish(true)
                .s3Bucket(bucketName)
                .s3Key(key)
                .build();

           UpdateFunctionCodeResponse response =
awsLambda.updateFunctionCode(functionCodeRequest);
           GetFunctionConfigurationRequest getFunctionConfigRequest =
GetFunctionConfigurationRequest.builder()
                .functionName(functionName)
```

```
                .build();

            WaiterResponse<GetFunctionConfigurationResponse> waiterResponse =
 waiter
                .waitUntilFunctionUpdated(getFunctionConfigRequest);
            waiterResponse.matched().response().ifPresent(System.out::println);
            System.out.println("The last modified value is " +
 response.lastModified());

        } catch (LambdaException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
```

- Untuk detail API, lihat UpdateFunctionCode di AWS SDK for Java 2.x Referensi API.

JavaScript

SDK untuk JavaScript (V3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
const updateFunctionCode = async (funcName, newFunc) => {
  const client = new LambdaClient({});
  const code = await readFile(`${dirname}../functions/${newFunc}.zip`);
  const command = new UpdateFunctionCodeCommand({
    ZipFile: code,
    FunctionName: funcName,
    Architectures: [Architecture.arm64],
    Handler: "index.handler", // Required when sending a .zip file
    PackageType: PackageType.Zip, // Required when sending a .zip file
    Runtime: Runtime.nodejs16x, // Required when sending a .zip file
  });

  return client.send(command);
};
```

- Untuk detail API, lihat UpdateFunctionCode di AWS SDK untuk JavaScript Referensi API.

PHP

SDK untuk PHP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
public function updateFunctionCode($functionName, $s3Bucket, $s3Key)
{
    return $this->lambdaClient->updateFunctionCode([
        'FunctionName' => $functionName,
        'S3Bucket' => $s3Bucket,
        'S3Key' => $s3Key,
    ]);
}
```

- Untuk detail API, lihat UpdateFunctionCode di AWS SDK untuk PHP Referensi API.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Memperbarui fungsi bernama "MyFunctiondengan konten baru yang terkandung dalam file zip yang ditentukan. Untuk fungsi C#.NET Core Lambda file zip harus berisi rakitan yang dikompilasi.

```
Update-LMFunctionCode -FunctionName MyFunction -ZipFilename .\UpdatedCode.zip
```

Contoh 2: Contoh ini mirip dengan yang sebelumnya tetapi menggunakan objek Amazon S3 yang berisi kode yang diperbarui untuk memperbarui fungsi.

```
Update-LMFunctionCode -FunctionName MyFunction -BucketName amzn-s3-demo-bucket -
Key UpdatedCode.zip
```

- Untuk detail API, lihat UpdateFunctionCode di Alat AWS untuk PowerShell Referensi Cmdlet (V4).

Alat untuk PowerShell V5

Contoh 1: Memperbarui fungsi bernama "MyFunctiondengan konten baru yang terkandung dalam file zip yang ditentukan. Untuk fungsi C#.NET Core Lambda file zip harus berisi rakitan yang dikompilasi.

```
Update-LMFunctionCode -FunctionName MyFunction -ZipFilename .\UpdatedCode.zip
```

Contoh 2: Contoh ini mirip dengan yang sebelumnya tetapi menggunakan objek Amazon S3 yang berisi kode yang diperbarui untuk memperbarui fungsi.

```
Update-LMFunctionCode -FunctionName MyFunction -BucketName amzn-s3-demo-bucket -
Key UpdatedCode.zip
```

- Untuk detail API, lihat UpdateFunctionCode di Alat AWS untuk PowerShell Referensi Cmdlet (V5).

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
class LambdaWrapper:
    def __init__(self, lambda_client, iam_resource):
        self.lambda_client = lambda_client
        self.iam_resource = iam_resource
```

```
    def update_function_code(self, function_name, deployment_package):
        """
        Updates the code for a Lambda function by submitting a .zip archive that
contains
        the code for the function.

        :param function_name: The name of the function to update.
        :param deployment_package: The function code to update, packaged as bytes
in
                                   .zip format.
        :return: Data about the update, including the status.
        """
        try:
            response = self.lambda_client.update_function_code(
                FunctionName=function_name, ZipFile=deployment_package
            )
        except ClientError as err:
            logger.error(
                "Couldn't update function %s. Here's why: %s: %s",
                function_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response
```

- Untuk detail API, lihat UpdateFunctionCode di AWS Referensi API SDK untuk Python (Boto3).

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
class LambdaWrapper
  attr_accessor :lambda_client, :cloudwatch_client, :iam_client

  def initialize
    @lambda_client = Aws::Lambda::Client.new
    @cloudwatch_client = Aws::CloudWatchLogs::Client.new(region: 'us-east-1')
    @iam_client = Aws::IAM::Client.new(region: 'us-east-1')
    @logger = Logger.new($stdout)
    @logger.level = Logger::WARN
  end

  # Updates the code for a Lambda function by submitting a .zip archive that
 contains
  # the code for the function.
  #
  # @param function_name: The name of the function to update.
  # @param deployment_package: The function code to update, packaged as bytes in
  #                            .zip format.
  # @return: Data about the update, including the status.
  def update_function_code(function_name, deployment_package)
    @lambda_client.update_function_code(
      function_name: function_name,
      zip_file: deployment_package
    )
    @lambda_client.wait_until(:function_updated_v2, { function_name:
 function_name }) do |w|
      w.max_attempts = 5
      w.delay = 5
    end
  rescue Aws::Lambda::Errors::ServiceException => e
    @logger.error("There was an error updating function code for:
 #{function_name}:\n #{e.message}")
    nil
  rescue Aws::Waiters::Errors::WaiterFailed => e
    @logger.error("Failed waiting for #{function_name} to update:\n
 #{e.message}")
  end
```

- Untuk detail API, lihat UpdateFunctionCode di AWS SDK untuk Ruby Referensi API.

Rust

SDK for Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /** Given a Path to a zip file, update the function's code and wait for the
 update to finish. */
    pub async fn update_function_code(
        &self,
        zip_file: PathBuf,
        key: String,
    ) -> Result<UpdateFunctionCodeOutput, anyhow::Error> {
        let function_code = self.prepare_function(zip_file, Some(key)).await?;

        info!("Updating code for {}", self.lambda_name);
        let update = self
            .lambda_client
            .update_function_code()
            .function_name(self.lambda_name.clone())
            .s3_bucket(self.bucket.clone())
            .s3_key(function_code.s3_key().unwrap().to_string())
            .send()
            .await
            .map_err(anyhow::Error::from)?;

        self.wait_for_function_ready().await?;

        Ok(update)
    }

    /**
     * Upload function code from a path to a zip file.
     * The zip file must have an AL2 Linux-compatible binary called `bootstrap`.
     * The easiest way to create such a zip is to use `cargo lambda build --
output-format Zip`.
     */
    async fn prepare_function(
```

```
        &self,
        zip_file: PathBuf,
        key: Option<String>,
    ) -> Result<FunctionCode, anyhow::Error> {
        let body = ByteStream::from_path(zip_file).await?;

        let key = key.unwrap_or_else(|| format!("{}_code", self.lambda_name));

        info!("Uploading function code to s3://{}/{}", self.bucket, key);
        let _ = self
            .s3_client
            .put_object()
            .bucket(self.bucket.clone())
            .key(key.clone())
            .body(body)
            .send()
            .await?;

        Ok(FunctionCode::builder()
            .s3_bucket(self.bucket.clone())
            .s3_key(key)
            .build())
    }
```

- Untuk detail API, lihat UpdateFunctionCode di AWS Referensi SDK untuk Rust API.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    TRY.
        oo_result = lo_lmd->updatefunctioncode(     " oo_result is returned for
testing purposes. "
              iv_functionname = iv_function_name
              iv_zipfile = io_zip_file ).
```

```
        MESSAGE 'Lambda function code updated.' TYPE 'I'.
      CATCH /aws1/cx_lmdcodesigningcfgno00.
        MESSAGE 'Code signing configuration does not exist.' TYPE 'E'.
      CATCH /aws1/cx_lmdcodestorageexcdex.
        MESSAGE 'Maximum total code size per account exceeded.' TYPE 'E'.
      CATCH /aws1/cx_lmdcodeverification00.
        MESSAGE 'Code signature failed one or more validation checks for
signature mismatch or expiration.' TYPE 'E'.
      CATCH /aws1/cx_lmdinvalidcodesigex.
        MESSAGE 'Code signature failed the integrity check.' TYPE 'E'.
      CATCH /aws1/cx_lmdinvparamvalueex.
        MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
      CATCH /aws1/cx_lmdresourceconflictex.
        MESSAGE 'Resource already exists or another operation is in progress.'
TYPE 'E'.
      CATCH /aws1/cx_lmdresourcenotfoundex.
        MESSAGE 'The requested resource does not exist.' TYPE 'E'.
      CATCH /aws1/cx_lmdserviceexception.
        MESSAGE 'An internal problem was encountered by the AWS Lambda service.'
TYPE 'E'.
      CATCH /aws1/cx_lmdtoomanyrequestsex.
        MESSAGE 'The maximum request throughput was reached.' TYPE 'E'.
    ENDTRY.
```

- Untuk detail API, lihat UpdateFunctionCode di AWS SDK untuk referensi API SAP ABAP.

Swift

SDK para Swift

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
import AWSClientRuntime
import AWSLambda
import Foundation
```

```
        let zipUrl = URL(fileURLWithPath: path)
        let zipData: Data

        // Read the function's Zip file.

        do {
            zipData = try Data(contentsOf: zipUrl)
        } catch {
            throw ExampleError.zipFileReadError
        }

        // Update the function's code and wait for the updated version to be
        // ready for use.

        do {
            _ = try await lambdaClient.updateFunctionCode(
                input: UpdateFunctionCodeInput(
                    functionName: functionName,
                    zipFile: zipData
                )
            )
        } catch {
            return false
        }
```

- Untuk detail API, lihat UpdateFunctionCode di AWS SDK untuk referensi API Swift.

Untuk daftar lengkap AWS Panduan pengembang SDK dan contoh kode, lihat Menggunakan Lambda dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan **`UpdateFunctionConfiguration`** dengan AWS SDK atau CLI

Contoh kode berikut menunjukkan cara menggunakan `UpdateFunctionConfiguration`.

Contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Anda dapat melihat tindakan ini dalam konteks dalam contoh kode berikut:

- Pelajari dasar-dasarnya

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Update the code of a Lambda function.
    /// </summary>
    /// <param name="functionName">The name of the function to update.</param>
    /// <param name="functionHandler">The code that performs the function's
actions.</param>
    /// <param name="environmentVariables">A dictionary of environment
variables.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> UpdateFunctionConfigurationAsync(
        string functionName,
        string functionHandler,
        Dictionary<string, string> environmentVariables)
    {
        var request = new UpdateFunctionConfigurationRequest
        {
            Handler = functionHandler,
            FunctionName = functionName,
            Environment = new Amazon.Lambda.Model.Environment { Variables =
environmentVariables },
        };

        var response = await
_lambdaService.UpdateFunctionConfigurationAsync(request);

        Console.WriteLine(response.LastModified);

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- Untuk detail API, lihat UpdateFunctionConfiguration di AWS SDK untuk .NET Referensi API.

C++

SDK untuk C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
       Aws::Client::ClientConfiguration clientConfig;
       // Optional: Set to the AWS Region in which the bucket was created
(overrides config file).
       // clientConfig.region = "us-east-1";

   Aws::Lambda::LambdaClient client(clientConfig);

       Aws::Lambda::Model::UpdateFunctionConfigurationRequest request;
       request.SetFunctionName(LAMBDA_NAME);
       Aws::Lambda::Model::Environment environment;
       environment.AddVariables("LOG_LEVEL", "DEBUG");
       request.SetEnvironment(environment);

       Aws::Lambda::Model::UpdateFunctionConfigurationOutcome outcome =
client.UpdateFunctionConfiguration(
               request);

       if (outcome.IsSuccess()) {
           std::cout << "The lambda configuration was successfully updated."
                     << std::endl;
           break;
       }

       else {
           std::cerr << "Error with Lambda::UpdateFunctionConfiguration. "
                     << outcome.GetError().GetMessage()
                     << std::endl;
       }
```

- Untuk detail API, lihat UpdateFunctionConfiguration di AWS SDK untuk C++ Referensi API.

CLI

AWS CLI

Untuk memodifikasi konfigurasi fungsi

Berikut ini `update-function-configuration` contoh memodifikasi ukuran memori menjadi 256 MB untuk versi yang tidak dipublikasikan ($LATEST) dari `my-function` fungsi.

```
aws lambda update-function-configuration \
    --function-name  my-function \
    --memory-size 256
```

Output:

```
{
    "FunctionName": "my-function",
    "LastModified": "2019-09-26T20:28:40.438+0000",
    "RevisionId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
    "MemorySize": 256,
    "Version": "$LATEST",
    "Role": "arn:aws:iam::123456789012:role/service-role/my-function-role-
uy3l9qyq",
    "Timeout": 3,
    "Runtime": "nodejs10.x",
    "TracingConfig": {
        "Mode": "PassThrough"
    },
    "CodeSha256": "5tT2qgzYUHaqwR716pZ2dpkn/0J1FrzJmlKidWoaCgk=",
    "Description": "",
    "VpcConfig": {
        "SubnetIds": [],
        "VpcId": "",
        "SecurityGroupIds": []
    },
    "CodeSize": 304,
    "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-function",
    "Handler": "index.handler"
}
```

Untuk informasi lebih lanjut, lihat AWS Konfigurasi Fungsi Lambda di AWS Panduan Pengembang Lambda.

- Untuk detail API, lihat UpdateFunctionConfiguration di AWS CLI Referensi Perintah.

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
import (
 "bytes"
 "context"
 "encoding/json"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/lambda"
 "github.com/aws/aws-sdk-go-v2/service/lambda/types"
)

// FunctionWrapper encapsulates function actions used in the examples.
// It contains an AWS Lambda service client that is used to perform user actions.
type FunctionWrapper struct {
 LambdaClient *lambda.Client
}



// UpdateFunctionConfiguration updates a map of environment variables configured
 for
// the Lambda function specified by functionName.
func (wrapper FunctionWrapper) UpdateFunctionConfiguration(ctx context.Context,
 functionName string, envVars map[string]string) {
```

```
 _, err := wrapper.LambdaClient.UpdateFunctionConfiguration(ctx,
 &lambda.UpdateFunctionConfigurationInput{
  FunctionName: aws.String(functionName),
  Environment:  &types.Environment{Variables: envVars},
 })
 if err != nil {
  log.Panicf("Couldn't update configuration for %v. Here's why: %v",
 functionName, err)
 }
}
```

- Untuk detail API, lihat UpdateFunctionConfiguration di AWS SDK untuk Go Referensi API.

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /**
     * Updates the configuration of an AWS Lambda function.
     *
     * @param awsLambda     the {@link LambdaClient} instance to use for the AWS
Lambda operation
     * @param functionName  the name of the AWS Lambda function to update
     * @param handler       the new handler for the AWS Lambda function
     *
     * @throws LambdaException if there is an error while updating the function
configuration
     */
    public static void updateFunctionConfiguration(LambdaClient awsLambda, String
functionName, String handler) {
        try {
            UpdateFunctionConfigurationRequest configurationRequest =
UpdateFunctionConfigurationRequest.builder()
                .functionName(functionName)
```

```
                .handler(handler)
                .runtime(Runtime.JAVA17)
                .build();

            awsLambda.updateFunctionConfiguration(configurationRequest);

        } catch (LambdaException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
```

- Untuk detail API, lihat UpdateFunctionConfiguration di AWS SDK for Java 2.x Referensi API.

JavaScript

SDK untuk JavaScript (V3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
const updateFunctionConfiguration = async (funcName) => {
  const client = new LambdaClient({});
  const config = readFileSync(`${dirname}../functions/config.json`).toString();
  const command = new UpdateFunctionConfigurationCommand({
    ...JSON.parse(config),
    FunctionName: funcName,
  });
  const result = await client.send(command);
  await waitForFunctionUpdated({ FunctionName: funcName });
  return result;
};
```

- Untuk detail API, lihat UpdateFunctionConfiguration di AWS SDK untuk JavaScript Referensi API.

PHP

SDK untuk PHP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    public function updateFunctionConfiguration($functionName, $handler,
$environment = '')
    {
        return $this->lambdaClient->updateFunctionConfiguration([
            'FunctionName' => $functionName,
            'Handler' => "$handler.lambda_handler",
            'Environment' => $environment,
        ]);
    }
```

- Untuk detail API, lihat UpdateFunctionConfiguration di AWS SDK untuk PHP Referensi API.

PowerShell

Alat untuk PowerShell V4

Contoh 1: Contoh ini memperbarui Konfigurasi Fungsi Lambda yang ada

```
Update-LMFunctionConfiguration -FunctionName "MylambdaFunction123" -Handler
 "lambda_function.launch_instance" -Timeout 600 -Environment_Variable
 @{ "envvar1"="value";"envvar2"="value" } -Role arn:aws:iam::123456789101:role/
service-role/lambda -DeadLetterConfig_TargetArn arn:aws:sns:us-east-1:
 123456789101:MyfirstTopic
```

- Untuk detail API, lihat UpdateFunctionConfiguration di Alat AWS untuk PowerShell Referensi Cmdlet (V4).

Alat untuk PowerShell V5

Contoh 1: Contoh ini memperbarui Konfigurasi Fungsi Lambda yang ada

```
Update-LMFunctionConfiguration -FunctionName "MylambdaFunction123" -Handler
 "lambda_function.launch_instance" -Timeout 600 -Environment_Variable
 @{ "envvar1"="value";"envvar2"="value" } -Role arn:aws:iam::123456789101:role/
service-role/lambda -DeadLetterConfig_TargetArn arn:aws:sns:us-east-1:
 123456789101:MyfirstTopic
```

- Untuk detail API, lihat UpdateFunctionConfiguration di Alat AWS untuk PowerShell Referensi Cmdlet (V5).

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
class LambdaWrapper:
    def __init__(self, lambda_client, iam_resource):
        self.lambda_client = lambda_client
        self.iam_resource = iam_resource


    def update_function_configuration(self, function_name, env_vars):
        """
        Updates the environment variables for a Lambda function.

        :param function_name: The name of the function to update.
        :param env_vars: A dict of environment variables to update.
        :return: Data about the update, including the status.
        """
        try:
            response = self.lambda_client.update_function_configuration(
                FunctionName=function_name, Environment={"Variables": env_vars}
            )
        except ClientError as err:
            logger.error(
                "Couldn't update function configuration %s. Here's why: %s: %s",
                function_name,
```

```
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response
```

- Untuk detail API, lihat UpdateFunctionConfiguration di AWS Referensi API SDK untuk Python (Boto3).

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
class LambdaWrapper
  attr_accessor :lambda_client, :cloudwatch_client, :iam_client

  def initialize
    @lambda_client = Aws::Lambda::Client.new
    @cloudwatch_client = Aws::CloudWatchLogs::Client.new(region: 'us-east-1')
    @iam_client = Aws::IAM::Client.new(region: 'us-east-1')
    @logger = Logger.new($stdout)
    @logger.level = Logger::WARN
  end

  # Updates the environment variables for a Lambda function.
  # @param function_name: The name of the function to update.
  # @param log_level: The log level of the function.
  # @return: Data about the update, including the status.
  def update_function_configuration(function_name, log_level)
    @lambda_client.update_function_configuration({
                                                   function_name: function_name,
                                                   environment: {
```

```
                                                    variables: {
                                                      'LOG_LEVEL' => log_level
                                                    }
                                                  }
                                                })
    @lambda_client.wait_until(:function_updated_v2, { function_name:
function_name }) do |w|
      w.max_attempts = 5
      w.delay = 5
    end
  rescue Aws::Lambda::Errors::ServiceException => e
    @logger.error("There was an error updating configurations for
#{function_name}:\n #{e.message}")
  rescue Aws::Waiters::Errors::WaiterFailed => e
    @logger.error("Failed waiting for #{function_name} to activate:\n
#{e.message}")
  end
```

- Untuk detail API, lihat UpdateFunctionConfiguration di AWS SDK untuk Ruby Referensi API.

Rust

SDK for Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /** Update the environment for a function. */
    pub async fn update_function_configuration(
        &self,
        environment: Environment,
    ) -> Result<UpdateFunctionConfigurationOutput, anyhow::Error> {
        info!(
            ?environment,
            "Updating environment for {}", self.lambda_name
        );
        let updated = self
            .lambda_client
```

```
            .update_function_configuration()
            .function_name(self.lambda_name.clone())
            .environment(environment)
            .send()
            .await
            .map_err(anyhow::Error::from)?;

        self.wait_for_function_ready().await?;

        Ok(updated)
    }
```

- Untuk detail API, lihat UpdateFunctionConfiguration di AWS Referensi SDK untuk Rust API.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    TRY.
        oo_result = lo_lmd->updatefunctionconfiguration(     " oo_result is
returned for testing purposes. "
              iv_functionname = iv_function_name
              iv_runtime = iv_runtime
              iv_description  = 'Updated Lambda function'
              iv_memorysize  = iv_memory_size ).

        MESSAGE 'Lambda function configuration/settings updated.' TYPE 'I'.
      CATCH /aws1/cx_lmdcodesigningcfgno00.
        MESSAGE 'Code signing configuration does not exist.' TYPE 'E'.
      CATCH /aws1/cx_lmdcodeverification00.
        MESSAGE 'Code signature failed one or more validation checks for
signature mismatch or expiration.' TYPE 'E'.
      CATCH /aws1/cx_lmdinvalidcodesigex.
        MESSAGE 'Code signature failed the integrity check.' TYPE 'E'.
      CATCH /aws1/cx_lmdinvparamvalueex.
```

```
        MESSAGE 'The request contains a non-valid parameter.' TYPE 'E'.
      CATCH /aws1/cx_lmdresourceconflictex.
        MESSAGE 'Resource already exists or another operation is in progress.'
TYPE 'E'.
      CATCH /aws1/cx_lmdresourcenotfoundex.
        MESSAGE 'The requested resource does not exist.' TYPE 'E'.
      CATCH /aws1/cx_lmdserviceexception.
        MESSAGE 'An internal problem was encountered by the AWS Lambda service.'
TYPE 'E'.
      CATCH /aws1/cx_lmdtoomanyrequestsex.
        MESSAGE 'The maximum request throughput was reached.' TYPE 'E'.
    ENDTRY.
```

- Untuk detail API, lihat UpdateFunctionConfiguration di AWS SDK untuk referensi API SAP ABAP.

Swift

SDK para Swift

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
import AWSClientRuntime
import AWSLambda
import Foundation

    /// Tell the server-side component to log debug output by setting its
    /// environment's `LOG_LEVEL` to `DEBUG`.
    ///
    /// - Parameters:
    ///   - lambdaClient: The `LambdaClient` to use.
    ///   - functionName: The name of the AWS Lambda function to enable debug
    ///     logging for.
    ///
    /// - Throws: `ExampleError.environmentResponseMissingError`,
    ///   `ExampleError.updateFunctionConfigurationError`,
```

```
    ///   `ExampleError.environmentVariablesMissingError`,
    ///   `ExampleError.logLevelIncorrectError`,
    ///   `ExampleError.updateFunctionConfigurationError`
    func enableDebugLogging(lambdaClient: LambdaClient, functionName: String)
async throws {
        let envVariables = [
            "LOG_LEVEL": "DEBUG"
        ]
        let environment = LambdaClientTypes.Environment(variables: envVariables)

        do {
            let output = try await lambdaClient.updateFunctionConfiguration(
                input: UpdateFunctionConfigurationInput(
                    environment: environment,
                    functionName: functionName
                )
            )

            guard let response = output.environment else {
                throw ExampleError.environmentResponseMissingError
            }

            if response.error != nil {
                throw ExampleError.updateFunctionConfigurationError
            }

            guard let retVariables = response.variables else {
                throw ExampleError.environmentVariablesMissingError
            }

            for envVar in retVariables {
                if envVar.key == "LOG_LEVEL" && envVar.value != "DEBUG" {
                    print("*** Log level is not set to DEBUG!")
                    throw ExampleError.logLevelIncorrectError
                }
            }
        } catch {
            throw ExampleError.updateFunctionConfigurationError
        }
    }
```

- Untuk detail API, lihat UpdateFunctionConfiguration di AWS SDK untuk referensi API Swift.

Untuk daftar lengkap AWS Panduan pengembang SDK dan contoh kode, lihat Menggunakan Lambda dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Skenario untuk Lambda menggunakan AWS SDK

Contoh kode berikut menunjukkan cara menerapkan skenario umum di Lambda dengan AWS SDK. Skenario ini menunjukkan cara menyelesaikan tugas tertentu dengan memanggil beberapa fungsi dalam Lambda atau dikombinasikan dengan yang lain Layanan AWS. Setiap skenario menyertakan tautan ke kode sumber lengkap, di mana Anda dapat menemukan instruksi tentang cara mengatur dan menjalankan kode.

Skenario menargetkan pengalaman tingkat menengah untuk membantu Anda memahami tindakan layanan dalam konteks.

Contoh

- Secara otomatis mengonfirmasi pengguna Amazon Cognito yang dikenal dengan fungsi Lambda menggunakan AWS SDK
- Secara otomatis memigrasikan pengguna Amazon Cognito yang dikenal dengan fungsi Lambda menggunakan AWS SDK
- Buat API Gateway REST API untuk melacak COVID-19 data
- Membuat API REST pustaka peminjaman
- Membuat aplikasi messenger dengan Step Functions
- Buat API rest dengan integrasi proxy fungsi
- Membuat aplikasi manajemen aset foto yang memungkinkan pengguna mengelola foto menggunakan label
- Membuat aplikasi obrolan websocket dengan API Gateway
- Buat aplikasi yang menganalisis umpan balik pelanggan dan mensintesis audio
- Membuat dasbor pemantauan dengan nama fungsi sebagai variabel
- Membuat fungsi tanpa server pertama Anda
- Menginvokasi fungsi Lambda dari browser
- Ubah data untuk aplikasi Anda dengan S3 Object Lambda
- Menggunakan API Gateway untuk menginvokasi fungsi Lambda
- Menggunakan Step Functions untuk menginvokasi fungsi Lambda

- Menggunakan peristiwa terjadwal untuk menginvokasi fungsi Lambda
- Gunakan Amazon Neptune API untuk mengembangkan fungsi Lambda yang menanyakan data grafik
- Menggunakan variabel properti dalam dasbor pemantauan untuk memantau beberapa fungsi tanpa server
- Tulis data aktivitas khusus dengan fungsi Lambda setelah otentikasi pengguna Amazon Cognito menggunakan AWS SDK

Secara otomatis mengonfirmasi pengguna Amazon Cognito yang dikenal dengan fungsi Lambda menggunakan AWS SDK

Contoh kode berikut menunjukkan cara mengonfirmasi secara otomatis pengguna Amazon Cognito yang dikenal dengan fungsi Lambda.

- Konfigurasikan kumpulan pengguna untuk memanggil fungsi Lambda untuk `PreSignUp` pemicu.
- Daftarkan pengguna dengan Amazon Cognito.
- Fungsi Lambda memindai tabel DynamoDB dan secara otomatis mengonfirmasi pengguna yang dikenal.
- Masuk sebagai pengguna baru, lalu bersihkan sumber daya.

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkap dan pelajari cara menyiapkan dan menjalankan di Repositori Contoh Kode AWS.

Jalankan skenario interaktif di penggugah/prompt perintah.

```
import (
 "context"
 "errors"
 "log"
```

```
 "strings"
 "user_pools_and_lambda_triggers/actions"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

// AutoConfirm separates the steps of this scenario into individual functions so
 that
// they are simpler to read and understand.
type AutoConfirm struct {
 helper       IScenarioHelper
 questioner   demotools.IQuestioner
 resources    Resources
 cognitoActor *actions.CognitoActions
}

// NewAutoConfirm constructs a new auto confirm runner.
func NewAutoConfirm(sdkConfig aws.Config, questioner demotools.IQuestioner,
 helper IScenarioHelper) AutoConfirm {
 scenario := AutoConfirm{
  helper:       helper,
  questioner:   questioner,
  resources:    Resources{},
  cognitoActor: &actions.CognitoActions{CognitoClient:
 cognitoidentityprovider.NewFromConfig(sdkConfig)},
 }
 scenario.resources.init(scenario.cognitoActor, questioner)
 return scenario
}

// AddPreSignUpTrigger adds a Lambda handler as an invocation target for the
 PreSignUp trigger.
func (runner *AutoConfirm) AddPreSignUpTrigger(ctx context.Context, userPoolId
 string, functionArn string) {
 log.Printf("Let's add a Lambda function to handle the PreSignUp trigger from
 Cognito.\n" +
  "This trigger happens when a user signs up, and lets your function take action
 before the main Cognito\n" +
  "sign up processing occurs.\n")
 err := runner.cognitoActor.UpdateTriggers(
  ctx, userPoolId,
```

```
  actions.TriggerInfo{Trigger: actions.PreSignUp, HandlerArn:
 aws.String(functionArn)})
 if err != nil {
  panic(err)
 }
 log.Printf("Lambda function %v added to user pool %v to handle the PreSignUp
 trigger.\n",
  functionArn, userPoolId)
}

// SignUpUser signs up a user from the known user table with a password you
 specify.
func (runner *AutoConfirm) SignUpUser(ctx context.Context, clientId string,
 usersTable string) (string, string) {
 log.Println("Let's sign up a user to your Cognito user pool. When the user's
 email matches an email in the\n" +
  "DynamoDB known users table, it is automatically verified and the user is
 confirmed.")

 knownUsers, err := runner.helper.GetKnownUsers(ctx, usersTable)
 if err != nil {
  panic(err)
 }
 userChoice := runner.questioner.AskChoice("Which user do you want to use?\n",
 knownUsers.UserNameList())
 user := knownUsers.Users[userChoice]

 var signedUp bool
 var userConfirmed bool
 password := runner.questioner.AskPassword("Enter a password that has at least
 eight characters, uppercase, lowercase, numbers and symbols.\n"+
  "(the password will not display as you type):", 8)
 for !signedUp {
  log.Printf("Signing up user '%v' with email '%v' to Cognito.\n", user.UserName,
 user.UserEmail)
  userConfirmed, err = runner.cognitoActor.SignUp(ctx, clientId, user.UserName,
 password, user.UserEmail)
  if err != nil {
   var invalidPassword *types.InvalidPasswordException
   if errors.As(err, &invalidPassword) {
    password = runner.questioner.AskPassword("Enter another password:", 8)
   } else {
    panic(err)
   }
```

```
  } else {
   signedUp = true
  }
 }
 log.Printf("User %v signed up, confirmed = %v.\n", user.UserName, userConfirmed)

 log.Println(strings.Repeat("-", 88))

 return user.UserName, password
}

// SignInUser signs in a user.
func (runner *AutoConfirm) SignInUser(ctx context.Context, clientId string,
 userName string, password string) string {
 runner.questioner.Ask("Press Enter when you're ready to continue.")
 log.Printf("Let's sign in as %v...\n", userName)
 authResult, err := runner.cognitoActor.SignIn(ctx, clientId, userName, password)
 if err != nil {
  panic(err)
 }
 log.Printf("Successfully signed in. Your access token starts with: %v...\n",
 (*authResult.AccessToken)[:10])
 log.Println(strings.Repeat("-", 88))
 return *authResult.AccessToken
}

// Run runs the scenario.
func (runner *AutoConfirm) Run(ctx context.Context, stackName string) {
 defer func() {
  if r := recover(); r != nil {
   log.Println("Something went wrong with the demo.")
   runner.resources.Cleanup(ctx)
  }
 }()

 log.Println(strings.Repeat("-", 88))
 log.Printf("Welcome\n")

 log.Println(strings.Repeat("-", 88))

 stackOutputs, err := runner.helper.GetStackOutputs(ctx, stackName)
 if err != nil {
  panic(err)
 }
```

```
 runner.resources.userPoolId = stackOutputs["UserPoolId"]
 runner.helper.PopulateUserTable(ctx, stackOutputs["TableName"])

 runner.AddPreSignUpTrigger(ctx, stackOutputs["UserPoolId"],
 stackOutputs["AutoConfirmFunctionArn"])
 runner.resources.triggers = append(runner.resources.triggers, actions.PreSignUp)
 userName, password := runner.SignUpUser(ctx, stackOutputs["UserPoolClientId"],
 stackOutputs["TableName"])
 runner.helper.ListRecentLogEvents(ctx, stackOutputs["AutoConfirmFunction"])
 runner.resources.userAccessTokens = append(runner.resources.userAccessTokens,
  runner.SignInUser(ctx, stackOutputs["UserPoolClientId"], userName, password))

 runner.resources.Cleanup(ctx)

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}
```

Tangani `PreSignUp` pemicu dengan fungsi Lambda.

```
import (
 "context"
 "log"
 "os"

 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 dynamodbtypes "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

const TABLE_NAME = "TABLE_NAME"

// UserInfo defines structured user data that can be marshalled to a DynamoDB
 format.
type UserInfo struct {
```

```
 UserName  string `dynamodbav:"UserName"`
 UserEmail string `dynamodbav:"UserEmail"`
}

// GetKey marshals the user email value to a DynamoDB key format.
func (user UserInfo) GetKey() map[string]dynamodbtypes.AttributeValue {
 userEmail, err := attributevalue.Marshal(user.UserEmail)
 if err != nil {
  panic(err)
 }
 return map[string]dynamodbtypes.AttributeValue{"UserEmail": userEmail}
}

type handler struct {
 dynamoClient *dynamodb.Client
}

// HandleRequest handles the PreSignUp event by looking up a user in an Amazon
 DynamoDB table and
// specifying whether they should be confirmed and verified.
func (h *handler) HandleRequest(ctx context.Context, event
 events.CognitoEventUserPoolsPreSignup) (events.CognitoEventUserPoolsPreSignup,
 error) {
 log.Printf("Received presignup from %v for user '%v'", event.TriggerSource,
 event.UserName)
 if event.TriggerSource != "PreSignUp_SignUp" {
  // Other trigger sources, such as PreSignUp_AdminInitiateAuth, ignore the
 response from this handler.
  return event, nil
 }
 tableName := os.Getenv(TABLE_NAME)
 user := UserInfo{
  UserEmail: event.Request.UserAttributes["email"],
 }
 log.Printf("Looking up email %v in table %v.\n", user.UserEmail, tableName)
 output, err := h.dynamoClient.GetItem(ctx, &dynamodb.GetItemInput{
  Key:       user.GetKey(),
  TableName: aws.String(tableName),
 })
 if err != nil {
  log.Printf("Error looking up email %v.\n", user.UserEmail)
  return event, err
 }
 if output.Item == nil {
```

```
  log.Printf("Email %v not found. Email verification is required.\n",
 user.UserEmail)
  return event, err
 }

 err = attributevalue.UnmarshalMap(output.Item, &user)
 if err != nil {
  log.Printf("Couldn't unmarshal DynamoDB item. Here's why: %v\n", err)
  return event, err
 }

 if user.UserName != event.UserName {
  log.Printf("UserEmail %v found, but stored UserName '%v' does not match
 supplied UserName '%v'. Verification is required.\n",
   user.UserEmail, user.UserName, event.UserName)
 } else {
  log.Printf("UserEmail %v found with matching UserName %v. User is confirmed.
\n", user.UserEmail, user.UserName)
  event.Response.AutoConfirmUser = true
  event.Response.AutoVerifyEmail = true
 }

 return event, err
}

func main() {
 ctx := context.Background()
 sdkConfig, err := config.LoadDefaultConfig(ctx)
 if err != nil {
  log.Panicln(err)
 }
 h := handler{
  dynamoClient: dynamodb.NewFromConfig(sdkConfig),
 }
 lambda.Start(h.HandleRequest)
}
```

Buat struct yang melakukan tugas-tugas umum.

```
import (
```

```
 "context"
 "log"
 "strings"
 "time"
 "user_pools_and_lambda_triggers/actions"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cloudformation"
 "github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

// IScenarioHelper defines common functions used by the workflows in this
 example.
type IScenarioHelper interface {
 Pause(secs int)
 GetStackOutputs(ctx context.Context, stackName string) (actions.StackOutputs,
 error)
 PopulateUserTable(ctx context.Context, tableName string)
 GetKnownUsers(ctx context.Context, tableName string) (actions.UserList, error)
 AddKnownUser(ctx context.Context, tableName string, user actions.User)
 ListRecentLogEvents(ctx context.Context, functionName string)
}

// ScenarioHelper contains AWS wrapper structs used by the workflows in this
 example.
type ScenarioHelper struct {
 questioner  demotools.IQuestioner
 dynamoActor *actions.DynamoActions
 cfnActor    *actions.CloudFormationActions
 cwlActor    *actions.CloudWatchLogsActions
 isTestRun   bool
}

// NewScenarioHelper constructs a new scenario helper.
func NewScenarioHelper(sdkConfig aws.Config, questioner demotools.IQuestioner)
 ScenarioHelper {
 scenario := ScenarioHelper{
  questioner:  questioner,
  dynamoActor: &actions.DynamoActions{DynamoClient:
 dynamodb.NewFromConfig(sdkConfig)},
  cfnActor:    &actions.CloudFormationActions{CfnClient:
 cloudformation.NewFromConfig(sdkConfig)},
```

```
  cwlActor:    &actions.CloudWatchLogsActions{CwlClient:
 cloudwatchlogs.NewFromConfig(sdkConfig)},
 }
 return scenario
}

// Pause waits for the specified number of seconds.
func (helper ScenarioHelper) Pause(secs int) {
 if !helper.isTestRun {
  time.Sleep(time.Duration(secs) * time.Second)
 }
}

// GetStackOutputs gets the outputs from the specified CloudFormation stack in a
 structured format.
func (helper ScenarioHelper) GetStackOutputs(ctx context.Context, stackName
 string) (actions.StackOutputs, error) {
 return helper.cfnActor.GetOutputs(ctx, stackName), nil
}

// PopulateUserTable fills the known user table with example data.
func (helper ScenarioHelper) PopulateUserTable(ctx context.Context, tableName
 string) {
 log.Printf("First, let's add some users to the DynamoDB %v table we'll use for
 this example.\n", tableName)
 err := helper.dynamoActor.PopulateTable(ctx, tableName)
 if err != nil {
  panic(err)
 }
}

// GetKnownUsers gets the users from the known users table in a structured
 format.
func (helper ScenarioHelper) GetKnownUsers(ctx context.Context, tableName string)
 (actions.UserList, error) {
 knownUsers, err := helper.dynamoActor.Scan(ctx, tableName)
 if err != nil {
  log.Printf("Couldn't get known users from table %v. Here's why: %v\n",
 tableName, err)
 }
 return knownUsers, err
}

// AddKnownUser adds a user to the known users table.
```

```
func (helper ScenarioHelper) AddKnownUser(ctx context.Context, tableName string,
 user actions.User) {
 log.Printf("Adding user '%v' with email '%v' to the DynamoDB known users
 table...\n",
  user.UserName, user.UserEmail)
 err := helper.dynamoActor.AddUser(ctx, tableName, user)
 if err != nil {
  panic(err)
 }
}

// ListRecentLogEvents gets the most recent log stream and events for the
 specified Lambda function and displays them.
func (helper ScenarioHelper) ListRecentLogEvents(ctx context.Context,
 functionName string) {
 log.Println("Waiting a few seconds to let Lambda write to CloudWatch Logs...")
 helper.Pause(10)
 log.Println("Okay, let's check the logs to find what's happened recently with
 your Lambda function.")
 logStream, err := helper.cwlActor.GetLatestLogStream(ctx, functionName)
 if err != nil {
  panic(err)
 }
 log.Printf("Getting some recent events from log stream %v\n",
 *logStream.LogStreamName)
 events, err := helper.cwlActor.GetLogEvents(ctx, functionName,
 *logStream.LogStreamName, 10)
 if err != nil {
  panic(err)
 }
 for _, event := range events {
  log.Printf("\t%v", *event.Message)
 }
 log.Println(strings.Repeat("-", 88))
}
```

Buat struct yang membungkus tindakan Amazon Cognito.

```
import (
 "context"
```

```
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
)

type CognitoActions struct {
 CognitoClient *cognitoidentityprovider.Client
}



// Trigger and TriggerInfo define typed data for updating an Amazon Cognito
 trigger.
type Trigger int

const (
 PreSignUp Trigger = iota
 UserMigration
 PostAuthentication
)

type TriggerInfo struct {
 Trigger    Trigger
 HandlerArn *string
}

// UpdateTriggers adds or removes Lambda triggers for a user pool. When a trigger
 is specified with a `nil` value,
// it is removed from the user pool.
func (actor CognitoActions) UpdateTriggers(ctx context.Context, userPoolId
 string, triggers ...TriggerInfo) error {
 output, err := actor.CognitoClient.DescribeUserPool(ctx,
 &cognitoidentityprovider.DescribeUserPoolInput{
  UserPoolId: aws.String(userPoolId),
 })
 if err != nil {
  log.Printf("Couldn't get info about user pool %v. Here's why: %v\n",
 userPoolId, err)
  return err
 }
 lambdaConfig := output.UserPool.LambdaConfig
```

```
 for _, trigger := range triggers {
  switch trigger.Trigger {
  case PreSignUp:
   lambdaConfig.PreSignUp = trigger.HandlerArn
  case UserMigration:
   lambdaConfig.UserMigration = trigger.HandlerArn
  case PostAuthentication:
   lambdaConfig.PostAuthentication = trigger.HandlerArn
  }
 }
 _, err = actor.CognitoClient.UpdateUserPool(ctx,
 &cognitoidentityprovider.UpdateUserPoolInput{
  UserPoolId:   aws.String(userPoolId),
  LambdaConfig: lambdaConfig,
 })
 if err != nil {
  log.Printf("Couldn't update user pool %v. Here's why: %v\n", userPoolId, err)
 }
 return err
}



// SignUp signs up a user with Amazon Cognito.
func (actor CognitoActions) SignUp(ctx context.Context, clientId string, userName
 string, password string, userEmail string) (bool, error) {
 confirmed := false
 output, err := actor.CognitoClient.SignUp(ctx,
 &cognitoidentityprovider.SignUpInput{
  ClientId: aws.String(clientId),
  Password: aws.String(password),
  Username: aws.String(userName),
  UserAttributes: []types.AttributeType{
   {Name: aws.String("email"), Value: aws.String(userEmail)},
  },
 })
 if err != nil {
  var invalidPassword *types.InvalidPasswordException
  if errors.As(err, &invalidPassword) {
   log.Println(*invalidPassword.Message)
  } else {
   log.Printf("Couldn't sign up user %v. Here's why: %v\n", userName, err)
  }
 } else {
```

```
  confirmed = output.UserConfirmed
 }
 return confirmed, err
}



// SignIn signs in a user to Amazon Cognito using a username and password
 authentication flow.
func (actor CognitoActions) SignIn(ctx context.Context, clientId string, userName
 string, password string) (*types.AuthenticationResultType, error) {
 var authResult *types.AuthenticationResultType
 output, err := actor.CognitoClient.InitiateAuth(ctx,
 &cognitoidentityprovider.InitiateAuthInput{
  AuthFlow:       "USER_PASSWORD_AUTH",
  ClientId:       aws.String(clientId),
  AuthParameters: map[string]string{"USERNAME": userName, "PASSWORD": password},
 })
 if err != nil {
  var resetRequired *types.PasswordResetRequiredException
  if errors.As(err, &resetRequired) {
   log.Println(*resetRequired.Message)
  } else {
   log.Printf("Couldn't sign in user %v. Here's why: %v\n", userName, err)
  }
 } else {
  authResult = output.AuthenticationResult
 }
 return authResult, err
}



// ForgotPassword starts a password recovery flow for a user. This flow typically
 sends a confirmation code
// to the user's configured notification destination, such as email.
func (actor CognitoActions) ForgotPassword(ctx context.Context, clientId string,
 userName string) (*types.CodeDeliveryDetailsType, error) {
 output, err := actor.CognitoClient.ForgotPassword(ctx,
 &cognitoidentityprovider.ForgotPasswordInput{
  ClientId: aws.String(clientId),
  Username: aws.String(userName),
 })
 if err != nil {
```

```
  log.Printf("Couldn't start password reset for user '%v'. Here;s why: %v\n",
 userName, err)
 }
 return output.CodeDeliveryDetails, err
}



// ConfirmForgotPassword confirms a user with a confirmation code and a new
 password.
func (actor CognitoActions) ConfirmForgotPassword(ctx context.Context, clientId
 string, code string, userName string, password string) error {
 _, err := actor.CognitoClient.ConfirmForgotPassword(ctx,
 &cognitoidentityprovider.ConfirmForgotPasswordInput{
  ClientId:         aws.String(clientId),
  ConfirmationCode: aws.String(code),
  Password:         aws.String(password),
  Username:         aws.String(userName),
 })
 if err != nil {
  var invalidPassword *types.InvalidPasswordException
  if errors.As(err, &invalidPassword) {
   log.Println(*invalidPassword.Message)
  } else {
   log.Printf("Couldn't confirm user %v. Here's why: %v", userName, err)
  }
 }
 return err
}



// DeleteUser removes a user from the user pool.
func (actor CognitoActions) DeleteUser(ctx context.Context, userAccessToken
 string) error {
 _, err := actor.CognitoClient.DeleteUser(ctx,
 &cognitoidentityprovider.DeleteUserInput{
  AccessToken: aws.String(userAccessToken),
 })
 if err != nil {
  log.Printf("Couldn't delete user. Here's why: %v\n", err)
 }
 return err
}
```

```
// AdminCreateUser uses administrator credentials to add a user to a user pool.
 This method leaves the user
// in a state that requires they enter a new password next time they sign in.
func (actor CognitoActions) AdminCreateUser(ctx context.Context, userPoolId
 string, userName string, userEmail string) error {
 _, err := actor.CognitoClient.AdminCreateUser(ctx,
 &cognitoidentityprovider.AdminCreateUserInput{
  UserPoolId:     aws.String(userPoolId),
  Username:       aws.String(userName),
  MessageAction:  types.MessageActionTypeSuppress,
  UserAttributes: []types.AttributeType{{Name: aws.String("email"), Value:
 aws.String(userEmail)}},
 })
 if err != nil {
  var userExists *types.UsernameExistsException
  if errors.As(err, &userExists) {
   log.Printf("User %v already exists in the user pool.", userName)
   err = nil
  } else {
   log.Printf("Couldn't create user %v. Here's why: %v\n", userName, err)
  }
 }
 return err
}



// AdminSetUserPassword uses administrator credentials to set a password for a
 user without requiring a
// temporary password.
func (actor CognitoActions) AdminSetUserPassword(ctx context.Context, userPoolId
 string, userName string, password string) error {
 _, err := actor.CognitoClient.AdminSetUserPassword(ctx,
 &cognitoidentityprovider.AdminSetUserPasswordInput{
  Password:   aws.String(password),
  UserPoolId: aws.String(userPoolId),
  Username:   aws.String(userName),
  Permanent:  true,
 })
 if err != nil {
  var invalidPassword *types.InvalidPasswordException
```

```
  if errors.As(err, &invalidPassword) {
   log.Println(*invalidPassword.Message)
  } else {
   log.Printf("Couldn't set password for user %v. Here's why: %v\n", userName,
 err)
  }
 }
 return err
}
```

Buat struct yang membungkus tindakan DynamoDB.

```
import (
 "context"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// DynamoActions encapsulates the Amazon Simple Notification Service (Amazon SNS)
 actions
// used in the examples.
type DynamoActions struct {
 DynamoClient *dynamodb.Client
}

// User defines structured user data.
type User struct {
 UserName  string
 UserEmail string
 LastLogin *LoginInfo `dynamodbav:",omitempty"`
}

// LoginInfo defines structured custom login data.
type LoginInfo struct {
 UserPoolId string
```

```
 ClientId   string
 Time       string
}

// UserList defines a list of users.
type UserList struct {
 Users []User
}

// UserNameList returns the usernames contained in a UserList as a list of
 strings.
func (users *UserList) UserNameList() []string {
 names := make([]string, len(users.Users))
 for i := 0; i < len(users.Users); i++ {
  names[i] = users.Users[i].UserName
 }
 return names
}

// PopulateTable adds a set of test users to the table.
func (actor DynamoActions) PopulateTable(ctx context.Context, tableName string)
 error {
 var err error
 var item map[string]types.AttributeValue
 var writeReqs []types.WriteRequest
 for i := 1; i < 4; i++ {
  item, err = attributevalue.MarshalMap(User{UserName: fmt.Sprintf("test_user_
%v", i), UserEmail: fmt.Sprintf("test_email_%v@example.com", i)})
  if err != nil {
   log.Printf("Couldn't marshall user into DynamoDB format. Here's why: %v\n",
 err)
   return err
  }
  writeReqs = append(writeReqs, types.WriteRequest{PutRequest:
 &types.PutRequest{Item: item}})
 }
 _, err = actor.DynamoClient.BatchWriteItem(ctx, &dynamodb.BatchWriteItemInput{
  RequestItems: map[string][]types.WriteRequest{tableName: writeReqs},
 })
 if err != nil {
  log.Printf("Couldn't populate table %v with users. Here's why: %v\n",
 tableName, err)
 }
 return err
```

```
}

// Scan scans the table for all items.
func (actor DynamoActions) Scan(ctx context.Context, tableName string) (UserList,
 error) {
 var userList UserList
 output, err := actor.DynamoClient.Scan(ctx, &dynamodb.ScanInput{
  TableName: aws.String(tableName),
 })
 if err != nil {
  log.Printf("Couldn't scan table %v for items. Here's why: %v\n", tableName,
 err)
 } else {
  err = attributevalue.UnmarshalListOfMaps(output.Items, &userList.Users)
  if err != nil {
   log.Printf("Couldn't unmarshal items into users. Here's why: %v\n", err)
  }
 }
 return userList, err
}

// AddUser adds a user item to a table.
func (actor DynamoActions) AddUser(ctx context.Context, tableName string, user
 User) error {
 userItem, err := attributevalue.MarshalMap(user)
 if err != nil {
  log.Printf("Couldn't marshall user to item. Here's why: %v\n", err)
 }
 _, err = actor.DynamoClient.PutItem(ctx, &dynamodb.PutItemInput{
  Item:      userItem,
  TableName: aws.String(tableName),
 })
 if err != nil {
  log.Printf("Couldn't put item in table %v. Here's why: %v", tableName, err)
 }
 return err
}
```

Buat struct yang membungkus CloudWatch Tindakan log.

```
import (
 "context"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs"
 "github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs/types"
)

type CloudWatchLogsActions struct {
 CwlClient *cloudwatchlogs.Client
}

// GetLatestLogStream gets the most recent log stream for a Lambda function.
func (actor CloudWatchLogsActions) GetLatestLogStream(ctx context.Context,
 functionName string) (types.LogStream, error) {
 var logStream types.LogStream
 logGroupName := fmt.Sprintf("/aws/lambda/%s", functionName)
 output, err := actor.CwlClient.DescribeLogStreams(ctx,
 &cloudwatchlogs.DescribeLogStreamsInput{
  Descending:   aws.Bool(true),
  Limit:        aws.Int32(1),
  LogGroupName: aws.String(logGroupName),
  OrderBy:      types.OrderByLastEventTime,
 })
 if err != nil {
  log.Printf("Couldn't get log streams for log group %v. Here's why: %v\n",
 logGroupName, err)
 } else {
  logStream = output.LogStreams[0]
 }
 return logStream, err
}

// GetLogEvents gets the most recent eventCount events from the specified log
 stream.
func (actor CloudWatchLogsActions) GetLogEvents(ctx context.Context, functionName
 string, logStreamName string, eventCount int32) (
 []types.OutputLogEvent, error) {
 var events []types.OutputLogEvent
 logGroupName := fmt.Sprintf("/aws/lambda/%s", functionName)
 output, err := actor.CwlClient.GetLogEvents(ctx,
 &cloudwatchlogs.GetLogEventsInput{
```

```
  LogStreamName: aws.String(logStreamName),
  Limit:         aws.Int32(eventCount),
  LogGroupName:  aws.String(logGroupName),
 })
 if err != nil {
  log.Printf("Couldn't get log event for log stream %v. Here's why: %v\n",
 logStreamName, err)
 } else {
  events = output.Events
 }
 return events, err
}
```

Buat struct yang membungkus CloudFormation tindakan.

```
import (
 "context"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cloudformation"
)

// StackOutputs defines a map of outputs from a specific stack.
type StackOutputs map[string]string

type CloudFormationActions struct {
 CfnClient *cloudformation.Client
}

// GetOutputs gets the outputs from a CloudFormation stack and puts them into a
 structured format.
func (actor CloudFormationActions) GetOutputs(ctx context.Context, stackName
 string) StackOutputs {
 output, err := actor.CfnClient.DescribeStacks(ctx,
 &cloudformation.DescribeStacksInput{
  StackName: aws.String(stackName),
 })
 if err != nil || len(output.Stacks) == 0 {
```

```
  log.Panicf("Couldn't find a CloudFormation stack named %v. Here's why: %v\n",
 stackName, err)
 }
 stackOutputs := StackOutputs{}
 for _, out := range output.Stacks[0].Outputs {
  stackOutputs[*out.OutputKey] = *out.OutputValue
 }
 return stackOutputs
}
```

Pembersihan sumber daya

```
import (
 "context"
 "log"
 "user_pools_and_lambda_triggers/actions"

 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

// Resources keeps track of AWS resources created during an example and handles
// cleanup when the example finishes.
type Resources struct {
 userPoolId       string
 userAccessTokens []string
 triggers         []actions.Trigger

 cognitoActor *actions.CognitoActions
 questioner   demotools.IQuestioner
}

func (resources *Resources) init(cognitoActor *actions.CognitoActions, questioner
 demotools.IQuestioner) {
 resources.userAccessTokens = []string{}
 resources.triggers = []actions.Trigger{}
 resources.cognitoActor = cognitoActor
 resources.questioner = questioner
}

// Cleanup deletes all AWS resources created during an example.
```

```
func (resources *Resources) Cleanup(ctx context.Context) {
 defer func() {
  if r := recover(); r != nil {
   log.Printf("Something went wrong during cleanup.\n%v\n", r)
   log.Println("Use the AWS Management Console to remove any remaining resources
 \n" +
    "that were created for this scenario.")
  }
 }()

 wantDelete := resources.questioner.AskBool("Do you want to remove all of the AWS
 resources that were created "+
  "during this demo (y/n)?", "y")
 if wantDelete {
  for _, accessToken := range resources.userAccessTokens {
   err := resources.cognitoActor.DeleteUser(ctx, accessToken)
   if err != nil {
    log.Println("Couldn't delete user during cleanup.")
    panic(err)
   }
   log.Println("Deleted user.")
  }
  triggerList := make([]actions.TriggerInfo, len(resources.triggers))
  for i := 0; i < len(resources.triggers); i++ {
   triggerList[i] = actions.TriggerInfo{Trigger: resources.triggers[i],
 HandlerArn: nil}
  }
  err := resources.cognitoActor.UpdateTriggers(ctx, resources.userPoolId,
 triggerList...)
  if err != nil {
   log.Println("Couldn't update Cognito triggers during cleanup.")
   panic(err)
  }
  log.Println("Removed Cognito triggers from user pool.")
 } else {
  log.Println("Be sure to remove resources when you're done with them to avoid
 unexpected charges!")
 }
}
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK untuk Go .

- DeleteUser
- InitiateAuth
- SignUp
- UpdateUserPool

JavaScript

SDK untuk JavaScript (V3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Konfigurasikan menjalankan “Skenario” interaktif. The JavaScript (v3) contoh berbagi skenario runner untuk merampingkan contoh kompleks. Kode sumber lengkap aktif GitHub.

```
import { AutoConfirm } from "./scenario-auto-confirm.js";

/**
 * The context is passed to every scenario. Scenario steps
 * will modify the context.
 */
const context = {
  errors: [],
  users: [
    {
      UserName: "test_user_1",
      UserEmail: "test_email_1@example.com",
    },
    {
      UserName: "test_user_2",
      UserEmail: "test_email_2@example.com",
    },
    {
      UserName: "test_user_3",
      UserEmail: "test_email_3@example.com",
    },
  ],
```

```
};

/**
 * Three Scenarios are created for the workflow. A Scenario is an orchestration
 class
 * that simplifies running a series of steps.
 */
export const scenarios = {
  // Demonstrate automatically confirming known users in a database.
  "auto-confirm": AutoConfirm(context),
};

// Call function if run directly
import { fileURLToPath } from "node:url";
import { parseScenarioArgs } from "@aws-doc-sdk-examples/lib/scenario/index.js";

if (process.argv[1] === fileURLToPath(import.meta.url)) {
  parseScenarioArgs(scenarios, {
    name: "Cognito user pools and triggers",
    description:
      "Demonstrate how to use the AWS SDKs to customize Amazon Cognito
 authentication behavior.",
  });
}
```

Skenario ini menunjukkan konfirmasi otomatis pengguna yang dikenal. Ini mengatur langkah-langkah contoh.

```
import { wait } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";
import {
  Scenario,
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/scenario.js";

import {
  getStackOutputs,
  logCleanUpReminder,
  promptForStackName,
  promptForStackRegion,
  skipWhenErrors,
```

```
} from "./steps-common.js";
import { populateTable } from "./actions/dynamodb-actions.js";
import {
  addPreSignUpHandler,
  deleteUser,
  getUser,
  signIn,
  signUpUser,
} from "./actions/cognito-actions.js";
import {
  getLatestLogStreamForLambda,
  getLogEvents,
} from "./actions/cloudwatch-logs-actions.js";

/**
 * @typedef {{
 *   errors: Error[],
 *   password: string,
 *   users: { UserName: string, UserEmail: string }[],
 *   selectedUser?: string,
 *   stackName?: string,
 *   stackRegion?: string,
 *   token?: string,
 *   confirmDeleteSignedInUser?: boolean,
 *   TableName?: string,
 *   UserPoolClientId?: string,
 *   UserPoolId?: string,
 *   UserPoolArn?: string,
 *   AutoConfirmHandlerArn?: string,
 *   AutoConfirmHandlerName?: string
 * }} State
 */

const greeting = new ScenarioOutput(
  "greeting",
  (/** @type {State} */ state) => `This demo will populate some users into the \
database created as part of the "${state.stackName}" stack. \
Then the AutoConfirmHandler will be linked to the PreSignUp \
trigger from Cognito. Finally, you will choose a user to sign up.`,
  { skipWhen: skipWhenErrors },
);

const logPopulatingUsers = new ScenarioOutput(
  "logPopulatingUsers",
```

```
  "Populating the DynamoDB table with some users.",
  { skipWhenErrors: skipWhenErrors },
);

const logPopulatingUsersComplete = new ScenarioOutput(
  "logPopulatingUsersComplete",
  "Done populating users.",
  { skipWhen: skipWhenErrors },
);

const populateUsers = new ScenarioAction(
  "populateUsers",
  async (/** @type {State} */ state) => {
    const [_, err] = await populateTable({
      region: state.stackRegion,
      tableName: state.TableName,
      items: state.users,
    });
    if (err) {
      state.errors.push(err);
    }
  },
  {
    skipWhen: skipWhenErrors,
  },
);

const logSetupSignUpTrigger = new ScenarioOutput(
  "logSetupSignUpTrigger",
  "Setting up the PreSignUp trigger for the Cognito User Pool.",
  { skipWhen: skipWhenErrors },
);

const setupSignUpTrigger = new ScenarioAction(
  "setupSignUpTrigger",
  async (/** @type {State} */ state) => {
    const [_, err] = await addPreSignUpHandler({
      region: state.stackRegion,
      userPoolId: state.UserPoolId,
      handlerArn: state.AutoConfirmHandlerArn,
    });
    if (err) {
      state.errors.push(err);
    }
```

```
  },
  {
    skipWhen: skipWhenErrors,
  },
);

const logSetupSignUpTriggerComplete = new ScenarioOutput(
  "logSetupSignUpTriggerComplete",
  (
    /** @type {State} */ state,
  ) => `The lambda function "${state.AutoConfirmHandlerName}" \
has been configured as the PreSignUp trigger handler for the user pool
 "${state.UserPoolId}".`,
  { skipWhen: skipWhenErrors },
);

const selectUser = new ScenarioInput(
  "selectedUser",
  "Select a user to sign up.",
  {
    type: "select",
    choices: (/** @type {State} */ state) => state.users.map((u) => u.UserName),
    skipWhen: skipWhenErrors,
    default: (/** @type {State} */ state) => state.users[0].UserName,
  },
);

const checkIfUserAlreadyExists = new ScenarioAction(
  "checkIfUserAlreadyExists",
  async (/** @type {State} */ state) => {
    const [user, err] = await getUser({
      region: state.stackRegion,
      userPoolId: state.UserPoolId,
      username: state.selectedUser,
    });

    if (err?.name === "UserNotFoundException") {
      // Do nothing. We're not expecting the user to exist before
      // sign up is complete.
      return;
    }

    if (err) {
      state.errors.push(err);
```

```
      return;
    }

    if (user) {
      state.errors.push(
        new Error(
          `The user "${state.selectedUser}" already exists in the user pool
 "${state.UserPoolId}".`,
        ),
      );
    }
  },
  {
    skipWhen: skipWhenErrors,
  },
);

const createPassword = new ScenarioInput(
  "password",
  "Enter a password that has at least eight characters, uppercase, lowercase,
 numbers and symbols.",
  { type: "password", skipWhen: skipWhenErrors, default: "Abcd1234!" },
);

const logSignUpExistingUser = new ScenarioOutput(
  "logSignUpExistingUser",
  (/** @type {State} */ state) => `Signing up user "${state.selectedUser}".`,
  { skipWhen: skipWhenErrors },
);

const signUpExistingUser = new ScenarioAction(
  "signUpExistingUser",
  async (/** @type {State} */ state) => {
    const signUp = (password) =>
      signUpUser({
        region: state.stackRegion,
        userPoolClientId: state.UserPoolClientId,
        username: state.selectedUser,
        email: state.users.find((u) => u.UserName === state.selectedUser)
          .UserEmail,
        password,
      });

    let [_, err] = await signUp(state.password);
```

```
    while (err?.name === "InvalidPasswordException") {
      console.warn("The password you entered was invalid.");
      await createPassword.handle(state);
      [_, err] = await signUp(state.password);
    }

    if (err) {
      state.errors.push(err);
    }
  },
  { skipWhen: skipWhenErrors },
);

const logSignUpExistingUserComplete = new ScenarioOutput(
  "logSignUpExistingUserComplete",
  (/** @type {State} */ state) =>
    `"${state.selectedUser} was signed up successfully.`,
  { skipWhen: skipWhenErrors },
);

const logLambdaLogs = new ScenarioAction(
  "logLambdaLogs",
  async (/** @type {State} */ state) => {
    console.log(
      "Waiting a few seconds to let Lambda write to CloudWatch Logs...\n",
    );
    await wait(10);

    const [logStream, logStreamErr] = await getLatestLogStreamForLambda({
      functionName: state.AutoConfirmHandlerName,
      region: state.stackRegion,
    });
    if (logStreamErr) {
      state.errors.push(logStreamErr);
      return;
    }

    console.log(
      `Getting some recent events from log stream "${logStream.logStreamName}"`,
    );
    const [logEvents, logEventsErr] = await getLogEvents({
      functionName: state.AutoConfirmHandlerName,
      region: state.stackRegion,
```

```
      eventCount: 10,
      logStreamName: logStream.logStreamName,
    });
    if (logEventsErr) {
      state.errors.push(logEventsErr);
      return;
    }

    console.log(logEvents.map((ev) => `\t${ev.message}`).join(""));
  },
  { skipWhen: skipWhenErrors },
);

const logSignInUser = new ScenarioOutput(
  "logSignInUser",
  (/** @type {State} */ state) => `Let's sign in as ${state.selectedUser}`,
  { skipWhen: skipWhenErrors },
);

const signInUser = new ScenarioAction(
  "signInUser",
  async (/** @type {State} */ state) => {
    const [response, err] = await signIn({
      region: state.stackRegion,
      clientId: state.UserPoolClientId,
      username: state.selectedUser,
      password: state.password,
    });

    if (err?.name === "PasswordResetRequiredException") {
      state.errors.push(new Error("Please reset your password."));
      return;
    }

    if (err) {
      state.errors.push(err);
      return;
    }

    state.token = response?.AuthenticationResult?.AccessToken;
  },
  { skipWhen: skipWhenErrors },
);
```

```
const logSignInUserComplete = new ScenarioOutput(
  "logSignInUserComplete",
  (/** @type {State} */ state) =>
    `Successfully signed in. Your access token starts with:
 ${state.token.slice(0, 11)}`,
  { skipWhen: skipWhenErrors },
);

const confirmDeleteSignedInUser = new ScenarioInput(
  "confirmDeleteSignedInUser",
  "Do you want to delete the currently signed in user?",
  { type: "confirm", skipWhen: skipWhenErrors },
);

const deleteSignedInUser = new ScenarioAction(
  "deleteSignedInUser",
  async (/** @type {State} */ state) => {
    const [_, err] = await deleteUser({
      region: state.stackRegion,
      accessToken: state.token,
    });

    if (err) {
      state.errors.push(err);
    }
  },
  {
    skipWhen: (/** @type {State} */ state) =>
      skipWhenErrors(state) || !state.confirmDeleteSignedInUser,
  },
);

const logErrors = new ScenarioOutput(
  "logErrors",
  (/** @type {State}*/ state) => {
    const errorList = state.errors
      .map((err) => ` - ${err.name}: ${err.message}`)
      .join("\n");
    return `Scenario errors found:\n${errorList}`;
  },
  {
    // Don't log errors when there aren't any!
    skipWhen: (/** @type {State} */ state) => state.errors.length === 0,
  },
```

```
);

export const AutoConfirm = (context) =>
  new Scenario(
    "AutoConfirm",
    [
      promptForStackName,
      promptForStackRegion,
      getStackOutputs,
      greeting,
      logPopulatingUsers,
      populateUsers,
      logPopulatingUsersComplete,
      logSetupSignUpTrigger,
      setupSignUpTrigger,
      logSetupSignUpTriggerComplete,
      selectUser,
      checkIfUserAlreadyExists,
      createPassword,
      logSignUpExistingUser,
      signUpExistingUser,
      logSignUpExistingUserComplete,
      logLambdaLogs,
      logSignInUser,
      signInUser,
      logSignInUserComplete,
      confirmDeleteSignedInUser,
      deleteSignedInUser,
      logCleanUpReminder,
      logErrors,
    ],
    context,
  );
```

Ini adalah langkah-langkah yang dibagikan dengan Skenario lain.

```
import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/scenario.js";
import { getCfnOutputs } from "@aws-doc-sdk-examples/lib/sdk/cfn-outputs.js";
```

```
export const skipWhenErrors = (state) => state.errors.length > 0;

export const getStackOutputs = new ScenarioAction(
  "getStackOutputs",
  async (state) => {
    if (!state.stackName || !state.stackRegion) {
      state.errors.push(
        new Error(
          "No stack name or region provided. The stack name and \
region are required to fetch CFN outputs relevant to this example.",
        ),
      );
      return;
    }

    const outputs = await getCfnOutputs(state.stackName, state.stackRegion);
    Object.assign(state, outputs);
  },
);

export const promptForStackName = new ScenarioInput(
  "stackName",
  "Enter the name of the stack you deployed earlier.",
  { type: "input", default: "PoolsAndTriggersStack" },
);

export const promptForStackRegion = new ScenarioInput(
  "stackRegion",
  "Enter the region of the stack you deployed earlier.",
  { type: "input", default: "us-east-1" },
);

export const logCleanUpReminder = new ScenarioOutput(
  "logCleanUpReminder",
  "All done. Remember to run 'cdk destroy' to teardown the stack.",
  { skipWhen: skipWhenErrors },
);
```

Seorang handler untuk `PreSignUp` pemicu dengan fungsi Lambda.

```
import type { PreSignUpTriggerEvent, Handler } from "aws-lambda";
```

```
import type { UserRepository } from "./user-repository";
import { DynamoDBUserRepository } from "./user-repository";

export class PreSignUpHandler {
  private userRepository: UserRepository;

  constructor(userRepository: UserRepository) {
    this.userRepository = userRepository;
  }

  private isPreSignUpTriggerSource(event: PreSignUpTriggerEvent): boolean {
    return event.triggerSource === "PreSignUp_SignUp";
  }

  private getEventUserEmail(event: PreSignUpTriggerEvent): string {
    return event.request.userAttributes.email;
  }

  async handlePreSignUpTriggerEvent(
    event: PreSignUpTriggerEvent,
  ): Promise<PreSignUpTriggerEvent> {
    console.log(
      `Received presignup from ${event.triggerSource} for user
 '${event.userName}'`,
    );

    if (!this.isPreSignUpTriggerSource(event)) {
      return event;
    }

    const eventEmail = this.getEventUserEmail(event);
    console.log(`Looking up email ${eventEmail}.`);
    const storedUserInfo =
      await this.userRepository.getUserInfoByEmail(eventEmail);

    if (!storedUserInfo) {
      console.log(
        `Email ${eventEmail} not found. Email verification is required.`,
      );
      return event;
    }

    if (storedUserInfo.UserName !== event.userName) {
      console.log(
```

```
        `UserEmail ${eventEmail} found, but stored UserName
 '${storedUserInfo.UserName}' does not match supplied UserName
 '${event.userName}'. Verification is required.`,
      );
    } else {
      console.log(
        `UserEmail ${eventEmail} found with matching UserName
 ${storedUserInfo.UserName}. User is confirmed.`,
      );
      event.response.autoConfirmUser = true;
      event.response.autoVerifyEmail = true;
    }
    return event;
  }
}

const createPreSignUpHandler = (): PreSignUpHandler => {
  const tableName = process.env.TABLE_NAME;
  if (!tableName) {
    throw new Error("TABLE_NAME environment variable is not set");
  }

  const userRepository = new DynamoDBUserRepository(tableName);
  return new PreSignUpHandler(userRepository);
};

export const handler: Handler = async (event: PreSignUpTriggerEvent) => {
  const preSignUpHandler = createPreSignUpHandler();
  return preSignUpHandler.handlePreSignUpTriggerEvent(event);
};
```

Modul dari CloudWatch Tindakan log.

```
import {
  CloudWatchLogsClient,
  GetLogEventsCommand,
  OrderBy,
  paginateDescribeLogStreams,
} from "@aws-sdk/client-cloudwatch-logs";

/**
```

```
 * Get the latest log stream for a Lambda function.
 * @param {{ functionName: string, region: string }} config
 * @returns {Promise<[import("@aws-sdk/client-cloudwatch-logs").LogStream | null,
 unknown]>}
 */
export const getLatestLogStreamForLambda = async ({ functionName, region }) => {
  try {
    const logGroupName = `/aws/lambda/${functionName}`;
    const cwlClient = new CloudWatchLogsClient({ region });
    const paginator = paginateDescribeLogStreams(
      { client: cwlClient },
      {
        descending: true,
        limit: 1,
        orderBy: OrderBy.LastEventTime,
        logGroupName,
      },
    );

    for await (const page of paginator) {
      return [page.logStreams[0], null];
    }
  } catch (err) {
    return [null, err];
  }
};

/**
 * Get the log events for a Lambda function's log stream.
 * @param {{
 *   functionName: string,
 *   logStreamName: string,
 *   eventCount: number,
 *   region: string
 * }} config
 * @returns {Promise<[import("@aws-sdk/client-cloudwatch-logs").OutputLogEvent[]
 | null, unknown]>}
 */
export const getLogEvents = async ({
  functionName,
  logStreamName,
  eventCount,
  region,
}) => {
```

```
  try {
    const cwlClient = new CloudWatchLogsClient({ region });
    const logGroupName = `/aws/lambda/${functionName}`;
    const response = await cwlClient.send(
      new GetLogEventsCommand({
        logStreamName: logStreamName,
        limit: eventCount,
        logGroupName: logGroupName,
      }),
    );

    return [response.events, null];
  } catch (err) {
    return [null, err];
  }
};
```

Modul tindakan Amazon Cognito.

```
import {
  AdminGetUserCommand,
  CognitoIdentityProviderClient,
  DeleteUserCommand,
  InitiateAuthCommand,
  SignUpCommand,
  UpdateUserPoolCommand,
} from "@aws-sdk/client-cognito-identity-provider";

/**
 * Connect a Lambda function to the PreSignUp trigger for a Cognito user pool
 * @param {{ region: string, userPoolId: string, handlerArn: string }} config
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-
provider").UpdateUserPoolCommandOutput | null, unknown]>}
 */
export const addPreSignUpHandler = async ({
  region,
  userPoolId,
  handlerArn,
}) => {
  try {
    const cognitoClient = new CognitoIdentityProviderClient({
```

```
      region,
    });

    const command = new UpdateUserPoolCommand({
      UserPoolId: userPoolId,
      LambdaConfig: {
        PreSignUp: handlerArn,
      },
    });

    const response = await cognitoClient.send(command);
    return [response, null];
  } catch (err) {
    return [null, err];
  }
};

/**
 * Attempt to register a user to a user pool with a given username and password.
 * @param {{
 *   region: string,
 *   userPoolClientId: string,
 *   username: string,
 *   email: string,
 *   password: string
 * }} config
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-
provider").SignUpCommandOutput | null, unknown]>}
 */
export const signUpUser = async ({
  region,
  userPoolClientId,
  username,
  email,
  password,
}) => {
  try {
    const cognitoClient = new CognitoIdentityProviderClient({
      region,
    });

    const response = await cognitoClient.send(
      new SignUpCommand({
        ClientId: userPoolClientId,
```

```
        Username: username,
        Password: password,
        UserAttributes: [{ Name: "email", Value: email }],
      }),
    );
    return [response, null];
  } catch (err) {
    return [null, err];
  }
};

/**
 * Sign in a user to Amazon Cognito using a username and password authentication
 flow.
 * @param {{ region: string, clientId: string, username: string, password:
 string }} config
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-
provider").InitiateAuthCommandOutput | null, unknown]>}
 */
export const signIn = async ({ region, clientId, username, password }) => {
  try {
    const cognitoClient = new CognitoIdentityProviderClient({ region });
    const response = await cognitoClient.send(
      new InitiateAuthCommand({
        AuthFlow: "USER_PASSWORD_AUTH",
        ClientId: clientId,
        AuthParameters: { USERNAME: username, PASSWORD: password },
      }),
    );
    return [response, null];
  } catch (err) {
    return [null, err];
  }
};

/**
 * Retrieve an existing user from a user pool.
 * @param {{ region: string, userPoolId: string, username: string }} config
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-
provider").AdminGetUserCommandOutput | null, unknown]>}
 */
export const getUser = async ({ region, userPoolId, username }) => {
  try {
    const cognitoClient = new CognitoIdentityProviderClient({ region });
```

```
    const response = await cognitoClient.send(
      new AdminGetUserCommand({
        UserPoolId: userPoolId,
        Username: username,
      }),
    );
    return [response, null];
  } catch (err) {
    return [null, err];
  }
};

/**
 * Delete the signed-in user. Useful for allowing a user to delete their
 * own profile.
 * @param {{ region: string, accessToken: string }} config
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-
provider").DeleteUserCommandOutput | null, unknown]>}
 */
export const deleteUser = async ({ region, accessToken }) => {
  try {
    const client = new CognitoIdentityProviderClient({ region });
    const response = await client.send(
      new DeleteUserCommand({ AccessToken: accessToken }),
    );
    return [response, null];
  } catch (err) {
    return [null, err];
  }
};
```

Modul tindakan DynamoDB.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import {
  BatchWriteCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";

/**
 * Populate a DynamoDB table with provide items.
```

```
 * @param {{ region: string, tableName: string, items: Record<string,
 unknown>[] }} config
 * @returns {Promise<[import("@aws-sdk/lib-dynamodb").BatchWriteCommandOutput |
 null, unknown]>}
 */
export const populateTable = async ({ region, tableName, items }) => {
  try {
    const ddbClient = new DynamoDBClient({ region });
    const docClient = DynamoDBDocumentClient.from(ddbClient);
    const response = await docClient.send(
      new BatchWriteCommand({
        RequestItems: {
          [tableName]: items.map((item) => ({
            PutRequest: {
              Item: item,
            },
          })),
        },
      }),
    );
    return [response, null];
  } catch (err) {
    return [null, err];
  }
};
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK untuk JavaScript .
  - DeleteUser
  - InitiateAuth
  - SignUp
  - UpdateUserPool

Untuk daftar lengkap AWS Panduan pengembang SDK dan contoh kode, lihat Menggunakan Lambda dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Secara otomatis memigrasikan pengguna Amazon Cognito yang dikenal dengan fungsi Lambda menggunakan AWS SDK

Contoh kode berikut menunjukkan cara memigrasi pengguna Amazon Cognito yang dikenal secara otomatis dengan fungsi Lambda.

- Konfigurasikan kumpulan pengguna untuk memanggil fungsi Lambda untuk `MigrateUser` pemicu.
- Masuk ke Amazon Cognito dengan nama pengguna dan email yang tidak ada di kumpulan pengguna.
- Fungsi Lambda memindai tabel DynamoDB dan secara otomatis memigrasikan pengguna yang dikenal ke kumpulan pengguna.
- Lakukan alur lupa kata sandi untuk mengatur ulang kata sandi untuk pengguna yang dimigrasi.
- Masuk sebagai pengguna baru, lalu bersihkan sumber daya.

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkap dan pelajari cara menyiapkan dan menjalankan di Repositori Contoh Kode AWS.

Jalankan skenario interaktif di penggugah/prompt perintah.

```
import (
 "context"
 "errors"
 "fmt"
 "log"
 "strings"
 "user_pools_and_lambda_triggers/actions"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
```

```
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

// MigrateUser separates the steps of this scenario into individual functions so
 that
// they are simpler to read and understand.
type MigrateUser struct {
 helper       IScenarioHelper
 questioner   demotools.IQuestioner
 resources    Resources
 cognitoActor *actions.CognitoActions
}

// NewMigrateUser constructs a new migrate user runner.
func NewMigrateUser(sdkConfig aws.Config, questioner demotools.IQuestioner,
 helper IScenarioHelper) MigrateUser {
 scenario := MigrateUser{
  helper:       helper,
  questioner:   questioner,
  resources:    Resources{},
  cognitoActor: &actions.CognitoActions{CognitoClient:
 cognitoidentityprovider.NewFromConfig(sdkConfig)},
 }
 scenario.resources.init(scenario.cognitoActor, questioner)
 return scenario
}

// AddMigrateUserTrigger adds a Lambda handler as an invocation target for the
 MigrateUser trigger.
func (runner *MigrateUser) AddMigrateUserTrigger(ctx context.Context, userPoolId
 string, functionArn string) {
 log.Printf("Let's add a Lambda function to handle the MigrateUser trigger from
 Cognito.\n" +
  "This trigger happens when an unknown user signs in, and lets your function
 take action before Cognito\n" +
  "rejects the user.\n\n")
 err := runner.cognitoActor.UpdateTriggers(
  ctx, userPoolId,
  actions.TriggerInfo{Trigger: actions.UserMigration, HandlerArn:
 aws.String(functionArn)})
 if err != nil {
  panic(err)
 }
```

```
 log.Printf("Lambda function %v added to user pool %v to handle the MigrateUser
 trigger.\n",
  functionArn, userPoolId)

 log.Println(strings.Repeat("-", 88))
}

// SignInUser adds a new user to the known users table and signs that user in to
 Amazon Cognito.
func (runner *MigrateUser) SignInUser(ctx context.Context, usersTable string,
 clientId string) (bool, actions.User) {
 log.Println("Let's sign in a user to your Cognito user pool. When the username
 and email matches an entry in the\n" +
  "DynamoDB known users table, the email is automatically verified and the user
 is migrated to the Cognito user pool.")

 user := actions.User{}
 user.UserName = runner.questioner.Ask("\nEnter a username:")
 user.UserEmail = runner.questioner.Ask("\nEnter an email that you own. This
 email will be used to confirm user migration\n" +
  "during this example:")

 runner.helper.AddKnownUser(ctx, usersTable, user)

 var err error
 var resetRequired *types.PasswordResetRequiredException
 var authResult *types.AuthenticationResultType
 signedIn := false
 for !signedIn && resetRequired == nil {
  log.Printf("Signing in to Cognito as user '%v'. The expected result is a
 PasswordResetRequiredException.\n\n", user.UserName)
  authResult, err = runner.cognitoActor.SignIn(ctx, clientId, user.UserName, "_")
  if err != nil {
   if errors.As(err, &resetRequired) {
    log.Printf("\nUser '%v' is not in the Cognito user pool but was found in the
 DynamoDB known users table.\n"+
     "User migration is started and a password reset is required.",
 user.UserName)
   } else {
    panic(err)
   }
  } else {
   log.Printf("User '%v' successfully signed in. This is unexpected and probably
 means you have not\n"+
```

```
    "cleaned up a previous run of this scenario, so the user exist in the Cognito
 user pool.\n"+
    "You can continue this example and select to clean up resources, or manually
 remove\n"+
    "the user from your user pool and try again.", user.UserName)
   runner.resources.userAccessTokens = append(runner.resources.userAccessTokens,
 *authResult.AccessToken)
   signedIn = true
  }
 }

 log.Println(strings.Repeat("-", 88))
 return resetRequired != nil, user
}

// ResetPassword starts a password recovery flow.
func (runner *MigrateUser) ResetPassword(ctx context.Context, clientId string,
 user actions.User) {
 wantCode := runner.questioner.AskBool(fmt.Sprintf("In order to migrate the user
 to Cognito, you must be able to receive a confirmation\n"+
  "code by email at %v. Do you want to send a code (y/n)?", user.UserEmail), "y")
 if !wantCode {
  log.Println("To complete this example and successfully migrate a user to
 Cognito, you must enter an email\n" +
   "you own that can receive a confirmation code.")
  return
 }
 codeDelivery, err := runner.cognitoActor.ForgotPassword(ctx, clientId,
 user.UserName)
 if err != nil {
  panic(err)
 }
 log.Printf("\nA confirmation code has been sent to %v.",
 *codeDelivery.Destination)
 code := runner.questioner.Ask("Check your email and enter it here:")

 confirmed := false
 password := runner.questioner.AskPassword("\nEnter a password that has at least
 eight characters, uppercase, lowercase, numbers and symbols.\n"+
  "(the password will not display as you type):", 8)
 for !confirmed {
  log.Printf("\nConfirming password reset for user '%v'.\n", user.UserName)
  err = runner.cognitoActor.ConfirmForgotPassword(ctx, clientId, code,
 user.UserName, password)
```

```
  if err != nil {
   var invalidPassword *types.InvalidPasswordException
   if errors.As(err, &invalidPassword) {
    password = runner.questioner.AskPassword("\nEnter another password:", 8)
   } else {
    panic(err)
   }
  } else {
   confirmed = true
  }
 }
 log.Printf("User '%v' successfully confirmed and migrated.\n", user.UserName)
 log.Println("Signing in with your username and password...")
 authResult, err := runner.cognitoActor.SignIn(ctx, clientId, user.UserName,
 password)
 if err != nil {
  panic(err)
 }
 log.Printf("Successfully signed in. Your access token starts with: %v...\n",
 (*authResult.AccessToken)[:10])
 runner.resources.userAccessTokens = append(runner.resources.userAccessTokens,
 *authResult.AccessToken)

 log.Println(strings.Repeat("-", 88))
}

// Run runs the scenario.
func (runner *MigrateUser) Run(ctx context.Context, stackName string) {
 defer func() {
  if r := recover(); r != nil {
   log.Println("Something went wrong with the demo.")
   runner.resources.Cleanup(ctx)
  }
 }()

 log.Println(strings.Repeat("-", 88))
 log.Printf("Welcome\n")

 log.Println(strings.Repeat("-", 88))

 stackOutputs, err := runner.helper.GetStackOutputs(ctx, stackName)
 if err != nil {
  panic(err)
 }
```

```
 runner.resources.userPoolId = stackOutputs["UserPoolId"]

 runner.AddMigrateUserTrigger(ctx, stackOutputs["UserPoolId"],
 stackOutputs["MigrateUserFunctionArn"])
 runner.resources.triggers = append(runner.resources.triggers,
 actions.UserMigration)
 resetNeeded, user := runner.SignInUser(ctx, stackOutputs["TableName"],
 stackOutputs["UserPoolClientId"])
 if resetNeeded {
  runner.helper.ListRecentLogEvents(ctx, stackOutputs["MigrateUserFunction"])
  runner.ResetPassword(ctx, stackOutputs["UserPoolClientId"], user)
 }

 runner.resources.Cleanup(ctx)

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}
```

Tangani `MigrateUser` pemicu dengan fungsi Lambda.

```
import (
 "context"
 "log"
 "os"

 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
)

const TABLE_NAME = "TABLE_NAME"

// UserInfo defines structured user data that can be marshalled to a DynamoDB
 format.
```

```
type UserInfo struct {
 UserName  string `dynamodbav:"UserName"`
 UserEmail string `dynamodbav:"UserEmail"`
}

type handler struct {
 dynamoClient *dynamodb.Client
}

// HandleRequest handles the MigrateUser event by looking up a user in an Amazon
 DynamoDB table and
// specifying whether they should be migrated to the user pool.
func (h *handler) HandleRequest(ctx context.Context, event
 events.CognitoEventUserPoolsMigrateUser)
 (events.CognitoEventUserPoolsMigrateUser, error) {
 log.Printf("Received migrate trigger from %v for user '%v'",
 event.TriggerSource, event.UserName)
 if event.TriggerSource != "UserMigration_Authentication" {
  return event, nil
 }
 tableName := os.Getenv(TABLE_NAME)
 user := UserInfo{
  UserName: event.UserName,
 }
 log.Printf("Looking up user '%v' in table %v.\n", user.UserName, tableName)
 filterEx := expression.Name("UserName").Equal(expression.Value(user.UserName))
 expr, err := expression.NewBuilder().WithFilter(filterEx).Build()
 if err != nil {
  log.Printf("Error building expression to query for user '%v'.\n",
 user.UserName)
  return event, err
 }
 output, err := h.dynamoClient.Scan(ctx, &dynamodb.ScanInput{
  TableName:                 aws.String(tableName),
  FilterExpression:          expr.Filter(),
  ExpressionAttributeNames:  expr.Names(),
  ExpressionAttributeValues: expr.Values(),
 })
 if err != nil {
  log.Printf("Error looking up user '%v'.\n", user.UserName)
  return event, err
 }
 if len(output.Items) == 0 {
  log.Printf("User '%v' not found, not migrating user.\n", user.UserName)
```

```
  return event, err
 }

 var users []UserInfo
 err = attributevalue.UnmarshalListOfMaps(output.Items, &users)
 if err != nil {
  log.Printf("Couldn't unmarshal DynamoDB items. Here's why: %v\n", err)
  return event, err
 }

 user = users[0]
 log.Printf("UserName '%v' found with email %v. User is migrated and must reset
 password.\n", user.UserName, user.UserEmail)
 event.CognitoEventUserPoolsMigrateUserResponse.UserAttributes =
 map[string]string{
  "email":          user.UserEmail,
  "email_verified": "true", // email_verified is required for the forgot password
 flow.
 }
 event.CognitoEventUserPoolsMigrateUserResponse.FinalUserStatus =
 "RESET_REQUIRED"
 event.CognitoEventUserPoolsMigrateUserResponse.MessageAction = "SUPPRESS"

 return event, err
}

func main() {
 ctx := context.Background()
 sdkConfig, err := config.LoadDefaultConfig(ctx)
 if err != nil {
  log.Panicln(err)
 }
 h := handler{
  dynamoClient: dynamodb.NewFromConfig(sdkConfig),
 }
 lambda.Start(h.HandleRequest)
}
```

Buat struct yang melakukan tugas-tugas umum.

```
import (
 "context"
 "log"
 "strings"
 "time"
 "user_pools_and_lambda_triggers/actions"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cloudformation"
 "github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

// IScenarioHelper defines common functions used by the workflows in this
 example.
type IScenarioHelper interface {
 Pause(secs int)
 GetStackOutputs(ctx context.Context, stackName string) (actions.StackOutputs,
 error)
 PopulateUserTable(ctx context.Context, tableName string)
 GetKnownUsers(ctx context.Context, tableName string) (actions.UserList, error)
 AddKnownUser(ctx context.Context, tableName string, user actions.User)
 ListRecentLogEvents(ctx context.Context, functionName string)
}

// ScenarioHelper contains AWS wrapper structs used by the workflows in this
 example.
type ScenarioHelper struct {
 questioner  demotools.IQuestioner
 dynamoActor *actions.DynamoActions
 cfnActor    *actions.CloudFormationActions
 cwlActor    *actions.CloudWatchLogsActions
 isTestRun   bool
}

// NewScenarioHelper constructs a new scenario helper.
func NewScenarioHelper(sdkConfig aws.Config, questioner demotools.IQuestioner)
 ScenarioHelper {
 scenario := ScenarioHelper{
  questioner:  questioner,
  dynamoActor: &actions.DynamoActions{DynamoClient:
 dynamodb.NewFromConfig(sdkConfig)},
```

```
  cfnActor:    &actions.CloudFormationActions{CfnClient:
 cloudformation.NewFromConfig(sdkConfig)},
  cwlActor:    &actions.CloudWatchLogsActions{CwlClient:
 cloudwatchlogs.NewFromConfig(sdkConfig)},
 }
 return scenario
}

// Pause waits for the specified number of seconds.
func (helper ScenarioHelper) Pause(secs int) {
 if !helper.isTestRun {
  time.Sleep(time.Duration(secs) * time.Second)
 }
}

// GetStackOutputs gets the outputs from the specified CloudFormation stack in a
 structured format.
func (helper ScenarioHelper) GetStackOutputs(ctx context.Context, stackName
 string) (actions.StackOutputs, error) {
 return helper.cfnActor.GetOutputs(ctx, stackName), nil
}

// PopulateUserTable fills the known user table with example data.
func (helper ScenarioHelper) PopulateUserTable(ctx context.Context, tableName
 string) {
 log.Printf("First, let's add some users to the DynamoDB %v table we'll use for
 this example.\n", tableName)
 err := helper.dynamoActor.PopulateTable(ctx, tableName)
 if err != nil {
  panic(err)
 }
}

// GetKnownUsers gets the users from the known users table in a structured
 format.
func (helper ScenarioHelper) GetKnownUsers(ctx context.Context, tableName string)
 (actions.UserList, error) {
 knownUsers, err := helper.dynamoActor.Scan(ctx, tableName)
 if err != nil {
  log.Printf("Couldn't get known users from table %v. Here's why: %v\n",
 tableName, err)
 }
 return knownUsers, err
}
```

```
// AddKnownUser adds a user to the known users table.
func (helper ScenarioHelper) AddKnownUser(ctx context.Context, tableName string,
 user actions.User) {
 log.Printf("Adding user '%v' with email '%v' to the DynamoDB known users
 table...\n",
  user.UserName, user.UserEmail)
 err := helper.dynamoActor.AddUser(ctx, tableName, user)
 if err != nil {
  panic(err)
 }
}

// ListRecentLogEvents gets the most recent log stream and events for the
 specified Lambda function and displays them.
func (helper ScenarioHelper) ListRecentLogEvents(ctx context.Context,
 functionName string) {
 log.Println("Waiting a few seconds to let Lambda write to CloudWatch Logs...")
 helper.Pause(10)
 log.Println("Okay, let's check the logs to find what's happened recently with
 your Lambda function.")
 logStream, err := helper.cwlActor.GetLatestLogStream(ctx, functionName)
 if err != nil {
  panic(err)
 }
 log.Printf("Getting some recent events from log stream %v\n",
 *logStream.LogStreamName)
 events, err := helper.cwlActor.GetLogEvents(ctx, functionName,
 *logStream.LogStreamName, 10)
 if err != nil {
  panic(err)
 }
 for _, event := range events {
  log.Printf("\t%v", *event.Message)
 }
 log.Println(strings.Repeat("-", 88))
}
```

Buat struct yang membungkus tindakan Amazon Cognito.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
)

type CognitoActions struct {
 CognitoClient *cognitoidentityprovider.Client
}



// Trigger and TriggerInfo define typed data for updating an Amazon Cognito
 trigger.
type Trigger int

const (
 PreSignUp Trigger = iota
 UserMigration
 PostAuthentication
)

type TriggerInfo struct {
 Trigger    Trigger
 HandlerArn *string
}

// UpdateTriggers adds or removes Lambda triggers for a user pool. When a trigger
 is specified with a `nil` value,
// it is removed from the user pool.
func (actor CognitoActions) UpdateTriggers(ctx context.Context, userPoolId
 string, triggers ...TriggerInfo) error {
 output, err := actor.CognitoClient.DescribeUserPool(ctx,
 &cognitoidentityprovider.DescribeUserPoolInput{
  UserPoolId: aws.String(userPoolId),
 })
 if err != nil {
  log.Printf("Couldn't get info about user pool %v. Here's why: %v\n",
 userPoolId, err)
  return err
```

```
 }
 lambdaConfig := output.UserPool.LambdaConfig
 for _, trigger := range triggers {
  switch trigger.Trigger {
  case PreSignUp:
   lambdaConfig.PreSignUp = trigger.HandlerArn
  case UserMigration:
   lambdaConfig.UserMigration = trigger.HandlerArn
  case PostAuthentication:
   lambdaConfig.PostAuthentication = trigger.HandlerArn
  }
 }
 _, err = actor.CognitoClient.UpdateUserPool(ctx,
 &cognitoidentityprovider.UpdateUserPoolInput{
  UserPoolId:   aws.String(userPoolId),
  LambdaConfig: lambdaConfig,
 })
 if err != nil {
  log.Printf("Couldn't update user pool %v. Here's why: %v\n", userPoolId, err)
 }
 return err
}



// SignUp signs up a user with Amazon Cognito.
func (actor CognitoActions) SignUp(ctx context.Context, clientId string, userName
 string, password string, userEmail string) (bool, error) {
 confirmed := false
 output, err := actor.CognitoClient.SignUp(ctx,
 &cognitoidentityprovider.SignUpInput{
  ClientId: aws.String(clientId),
  Password: aws.String(password),
  Username: aws.String(userName),
  UserAttributes: []types.AttributeType{
   {Name: aws.String("email"), Value: aws.String(userEmail)},
  },
 })
 if err != nil {
  var invalidPassword *types.InvalidPasswordException
  if errors.As(err, &invalidPassword) {
   log.Println(*invalidPassword.Message)
  } else {
   log.Printf("Couldn't sign up user %v. Here's why: %v\n", userName, err)
```

```
  }
 } else {
  confirmed = output.UserConfirmed
 }
 return confirmed, err
}



// SignIn signs in a user to Amazon Cognito using a username and password
 authentication flow.
func (actor CognitoActions) SignIn(ctx context.Context, clientId string, userName
 string, password string) (*types.AuthenticationResultType, error) {
 var authResult *types.AuthenticationResultType
 output, err := actor.CognitoClient.InitiateAuth(ctx,
 &cognitoidentityprovider.InitiateAuthInput{
  AuthFlow:       "USER_PASSWORD_AUTH",
  ClientId:       aws.String(clientId),
  AuthParameters: map[string]string{"USERNAME": userName, "PASSWORD": password},
 })
 if err != nil {
  var resetRequired *types.PasswordResetRequiredException
  if errors.As(err, &resetRequired) {
   log.Println(*resetRequired.Message)
  } else {
   log.Printf("Couldn't sign in user %v. Here's why: %v\n", userName, err)
  }
 } else {
  authResult = output.AuthenticationResult
 }
 return authResult, err
}



// ForgotPassword starts a password recovery flow for a user. This flow typically
 sends a confirmation code
// to the user's configured notification destination, such as email.
func (actor CognitoActions) ForgotPassword(ctx context.Context, clientId string,
 userName string) (*types.CodeDeliveryDetailsType, error) {
 output, err := actor.CognitoClient.ForgotPassword(ctx,
 &cognitoidentityprovider.ForgotPasswordInput{
  ClientId: aws.String(clientId),
  Username: aws.String(userName),
```

```
 })
 if err != nil {
  log.Printf("Couldn't start password reset for user '%v'. Here;s why: %v\n",
 userName, err)
 }
 return output.CodeDeliveryDetails, err
}



// ConfirmForgotPassword confirms a user with a confirmation code and a new
 password.
func (actor CognitoActions) ConfirmForgotPassword(ctx context.Context, clientId
 string, code string, userName string, password string) error {
 _, err := actor.CognitoClient.ConfirmForgotPassword(ctx,
 &cognitoidentityprovider.ConfirmForgotPasswordInput{
  ClientId:         aws.String(clientId),
  ConfirmationCode: aws.String(code),
  Password:         aws.String(password),
  Username:         aws.String(userName),
 })
 if err != nil {
  var invalidPassword *types.InvalidPasswordException
  if errors.As(err, &invalidPassword) {
   log.Println(*invalidPassword.Message)
  } else {
   log.Printf("Couldn't confirm user %v. Here's why: %v", userName, err)
  }
 }
 return err
}



// DeleteUser removes a user from the user pool.
func (actor CognitoActions) DeleteUser(ctx context.Context, userAccessToken
 string) error {
 _, err := actor.CognitoClient.DeleteUser(ctx,
 &cognitoidentityprovider.DeleteUserInput{
  AccessToken: aws.String(userAccessToken),
 })
 if err != nil {
  log.Printf("Couldn't delete user. Here's why: %v\n", err)
 }
```

```
 return err
}



// AdminCreateUser uses administrator credentials to add a user to a user pool.
 This method leaves the user
// in a state that requires they enter a new password next time they sign in.
func (actor CognitoActions) AdminCreateUser(ctx context.Context, userPoolId
 string, userName string, userEmail string) error {
 _, err := actor.CognitoClient.AdminCreateUser(ctx,
 &cognitoidentityprovider.AdminCreateUserInput{
  UserPoolId:     aws.String(userPoolId),
  Username:       aws.String(userName),
  MessageAction:  types.MessageActionTypeSuppress,
  UserAttributes: []types.AttributeType{{Name: aws.String("email"), Value:
 aws.String(userEmail)}},
 })
 if err != nil {
  var userExists *types.UsernameExistsException
  if errors.As(err, &userExists) {
   log.Printf("User %v already exists in the user pool.", userName)
   err = nil
  } else {
   log.Printf("Couldn't create user %v. Here's why: %v\n", userName, err)
  }
 }
 return err
}



// AdminSetUserPassword uses administrator credentials to set a password for a
 user without requiring a
// temporary password.
func (actor CognitoActions) AdminSetUserPassword(ctx context.Context, userPoolId
 string, userName string, password string) error {
 _, err := actor.CognitoClient.AdminSetUserPassword(ctx,
 &cognitoidentityprovider.AdminSetUserPasswordInput{
  Password:   aws.String(password),
  UserPoolId: aws.String(userPoolId),
  Username:   aws.String(userName),
  Permanent:  true,
 })
```

```
 if err != nil {
  var invalidPassword *types.InvalidPasswordException
  if errors.As(err, &invalidPassword) {
   log.Println(*invalidPassword.Message)
  } else {
   log.Printf("Couldn't set password for user %v. Here's why: %v\n", userName,
 err)
  }
 }
 return err
}
```

Buat struct yang membungkus tindakan DynamoDB.

```
import (
 "context"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// DynamoActions encapsulates the Amazon Simple Notification Service (Amazon SNS)
 actions
// used in the examples.
type DynamoActions struct {
 DynamoClient *dynamodb.Client
}

// User defines structured user data.
type User struct {
 UserName  string
 UserEmail string
 LastLogin *LoginInfo `dynamodbav:",omitempty"`
}

// LoginInfo defines structured custom login data.
```

```
type LoginInfo struct {
 UserPoolId string
 ClientId   string
 Time       string
}

// UserList defines a list of users.
type UserList struct {
 Users []User
}

// UserNameList returns the usernames contained in a UserList as a list of
 strings.
func (users *UserList) UserNameList() []string {
 names := make([]string, len(users.Users))
 for i := 0; i < len(users.Users); i++ {
  names[i] = users.Users[i].UserName
 }
 return names
}

// PopulateTable adds a set of test users to the table.
func (actor DynamoActions) PopulateTable(ctx context.Context, tableName string)
 error {
 var err error
 var item map[string]types.AttributeValue
 var writeReqs []types.WriteRequest
 for i := 1; i < 4; i++ {
  item, err = attributevalue.MarshalMap(User{UserName: fmt.Sprintf("test_user_
%v", i), UserEmail: fmt.Sprintf("test_email_%v@example.com", i)})
  if err != nil {
   log.Printf("Couldn't marshall user into DynamoDB format. Here's why: %v\n",
 err)
   return err
  }
  writeReqs = append(writeReqs, types.WriteRequest{PutRequest:
 &types.PutRequest{Item: item}})
 }
 _, err = actor.DynamoClient.BatchWriteItem(ctx, &dynamodb.BatchWriteItemInput{
  RequestItems: map[string][]types.WriteRequest{tableName: writeReqs},
 })
 if err != nil {
  log.Printf("Couldn't populate table %v with users. Here's why: %v\n",
 tableName, err)
```

```
 }
 return err
}

// Scan scans the table for all items.
func (actor DynamoActions) Scan(ctx context.Context, tableName string) (UserList,
 error) {
 var userList UserList
 output, err := actor.DynamoClient.Scan(ctx, &dynamodb.ScanInput{
  TableName: aws.String(tableName),
 })
 if err != nil {
  log.Printf("Couldn't scan table %v for items. Here's why: %v\n", tableName,
 err)
 } else {
  err = attributevalue.UnmarshalListOfMaps(output.Items, &userList.Users)
  if err != nil {
   log.Printf("Couldn't unmarshal items into users. Here's why: %v\n", err)
  }
 }
 return userList, err
}

// AddUser adds a user item to a table.
func (actor DynamoActions) AddUser(ctx context.Context, tableName string, user
 User) error {
 userItem, err := attributevalue.MarshalMap(user)
 if err != nil {
  log.Printf("Couldn't marshall user to item. Here's why: %v\n", err)
 }
 _, err = actor.DynamoClient.PutItem(ctx, &dynamodb.PutItemInput{
  Item:      userItem,
  TableName: aws.String(tableName),
 })
 if err != nil {
  log.Printf("Couldn't put item in table %v. Here's why: %v", tableName, err)
 }
 return err
}
```

Buat struct yang membungkus CloudWatch Tindakan log.

```
import (
 "context"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs"
 "github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs/types"
)

type CloudWatchLogsActions struct {
 CwlClient *cloudwatchlogs.Client
}

// GetLatestLogStream gets the most recent log stream for a Lambda function.
func (actor CloudWatchLogsActions) GetLatestLogStream(ctx context.Context,
 functionName string) (types.LogStream, error) {
 var logStream types.LogStream
 logGroupName := fmt.Sprintf("/aws/lambda/%s", functionName)
 output, err := actor.CwlClient.DescribeLogStreams(ctx,
 &cloudwatchlogs.DescribeLogStreamsInput{
  Descending:   aws.Bool(true),
  Limit:        aws.Int32(1),
  LogGroupName: aws.String(logGroupName),
  OrderBy:      types.OrderByLastEventTime,
 })
 if err != nil {
  log.Printf("Couldn't get log streams for log group %v. Here's why: %v\n",
 logGroupName, err)
 } else {
  logStream = output.LogStreams[0]
 }
 return logStream, err
}

// GetLogEvents gets the most recent eventCount events from the specified log
 stream.
func (actor CloudWatchLogsActions) GetLogEvents(ctx context.Context, functionName
 string, logStreamName string, eventCount int32) (
 []types.OutputLogEvent, error) {
 var events []types.OutputLogEvent
 logGroupName := fmt.Sprintf("/aws/lambda/%s", functionName)
```

```
 output, err := actor.CwlClient.GetLogEvents(ctx,
 &cloudwatchlogs.GetLogEventsInput{
  LogStreamName: aws.String(logStreamName),
  Limit:         aws.Int32(eventCount),
  LogGroupName:  aws.String(logGroupName),
 })
 if err != nil {
  log.Printf("Couldn't get log event for log stream %v. Here's why: %v\n",
 logStreamName, err)
 } else {
  events = output.Events
 }
 return events, err
}
```

Buat struct yang membungkus CloudFormation tindakan.

```
import (
 "context"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cloudformation"
)

// StackOutputs defines a map of outputs from a specific stack.
type StackOutputs map[string]string

type CloudFormationActions struct {
 CfnClient *cloudformation.Client
}

// GetOutputs gets the outputs from a CloudFormation stack and puts them into a
 structured format.
func (actor CloudFormationActions) GetOutputs(ctx context.Context, stackName
 string) StackOutputs {
 output, err := actor.CfnClient.DescribeStacks(ctx,
 &cloudformation.DescribeStacksInput{
  StackName: aws.String(stackName),
 })
```

```
 if err != nil || len(output.Stacks) == 0 {
  log.Panicf("Couldn't find a CloudFormation stack named %v. Here's why: %v\n",
 stackName, err)
 }
 stackOutputs := StackOutputs{}
 for _, out := range output.Stacks[0].Outputs {
  stackOutputs[*out.OutputKey] = *out.OutputValue
 }
 return stackOutputs
}
```

Pembersihan sumber daya

```
import (
 "context"
 "log"
 "user_pools_and_lambda_triggers/actions"

 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

// Resources keeps track of AWS resources created during an example and handles
// cleanup when the example finishes.
type Resources struct {
 userPoolId       string
 userAccessTokens []string
 triggers         []actions.Trigger

 cognitoActor *actions.CognitoActions
 questioner   demotools.IQuestioner
}

func (resources *Resources) init(cognitoActor *actions.CognitoActions, questioner
 demotools.IQuestioner) {
 resources.userAccessTokens = []string{}
 resources.triggers = []actions.Trigger{}
 resources.cognitoActor = cognitoActor
 resources.questioner = questioner
}
```

```
// Cleanup deletes all AWS resources created during an example.
func (resources *Resources) Cleanup(ctx context.Context) {
 defer func() {
  if r := recover(); r != nil {
   log.Printf("Something went wrong during cleanup.\n%v\n", r)
   log.Println("Use the AWS Management Console to remove any remaining resources
 \n" +
    "that were created for this scenario.")
  }
 }()

 wantDelete := resources.questioner.AskBool("Do you want to remove all of the AWS
 resources that were created "+
  "during this demo (y/n)?", "y")
 if wantDelete {
  for _, accessToken := range resources.userAccessTokens {
   err := resources.cognitoActor.DeleteUser(ctx, accessToken)
   if err != nil {
    log.Println("Couldn't delete user during cleanup.")
    panic(err)
   }
   log.Println("Deleted user.")
  }
  triggerList := make([]actions.TriggerInfo, len(resources.triggers))
  for i := 0; i < len(resources.triggers); i++ {
   triggerList[i] = actions.TriggerInfo{Trigger: resources.triggers[i],
 HandlerArn: nil}
  }
  err := resources.cognitoActor.UpdateTriggers(ctx, resources.userPoolId,
 triggerList...)
  if err != nil {
   log.Println("Couldn't update Cognito triggers during cleanup.")
   panic(err)
  }
  log.Println("Removed Cognito triggers from user pool.")
 } else {
  log.Println("Be sure to remove resources when you're done with them to avoid
 unexpected charges!")
 }
}
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK untuk Go .

- ConfirmForgotPassword
- DeleteUser
- ForgotPassword
- InitiateAuth
- SignUp
- UpdateUserPool

Untuk daftar lengkap AWS Panduan pengembang SDK dan contoh kode, lihat Menggunakan Lambda dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Buat API Gateway REST API untuk melacak COVID-19 data

Contoh kode berikut menunjukkan cara membuat REST API yang mensimulasikan sistem untuk melacak kasus harian COVID-19 di Amerika Serikat, menggunakan data fiksi.

Python

SDK untuk Python (Boto3)

Menunjukkan cara menggunakan AWS Piala dengan AWS SDK untuk Python (Boto3) untuk membuat REST API tanpa server yang menggunakan Amazon API Gateway, AWS Lambda, dan Amazon DynamoDB. REST API mensimulasikan sistem yang melacak kasus harian COVID-19 di Amerika Serikat, menggunakan data fiksi. Pelajari cara:

- Gunakan AWS Piala untuk menentukan rute dalam fungsi Lambda yang dipanggil untuk menangani permintaan REST yang datang melalui API Gateway.
- Menggunakan fungsi Lambda untuk mengambil dan menyimpan data dalam tabel DynamoDB untuk melayani permintaan REST.
- Tentukan struktur tabel dan sumber daya peran keamanan dalam AWS CloudFormation templat.
- Gunakan AWS Piala dan CloudFormation untuk mengemas dan menyebarkan semua sumber daya yang diperlukan.
- Gunakan CloudFormation untuk membersihkan semua sumber daya yang dibuat.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkap di GitHub.

Layanan yang digunakan dalam contoh ini

- API Gateway
- CloudFormation
- DynamoDB
- Lambda

Untuk daftar lengkap AWS Panduan pengembang SDK dan contoh kode, lihat Menggunakan Lambda dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Membuat API REST pustaka peminjaman

Contoh kode berikut menunjukkan cara membuat pustaka peminjaman tempat pelanggan dapat meminjam dan mengembalikan buku dengan menggunakan API REST yang didukung oleh basis data Amazon Aurora.

Python

SDK untuk Python (Boto3)

Menunjukkan cara menggunakan AWS SDK untuk Python (Boto3) dengan Amazon Relational Database Service (Amazon RDS) API dan AWS Piala untuk membuat REST API yang didukung oleh database Amazon Aurora. Layanan web sepenuhnya nirserver dan mewakili pustaka peminjaman sederhana tempat pelanggan dapat meminjam dan mengembalikan buku. Pelajari cara:

- Membuat dan mengelola klaster basis data Aurora nirserver.
- Gunakan AWS Secrets Manager untuk mengelola kredenSIAL database.
- Menerapkan lapisan penyimpanan data yang menggunakan Amazon RDS untuk memindahkan data masuk dan keluar dari basis data.
- Gunakan AWS Chalice untuk menerapkan REST API tanpa server ke Amazon API Gateway dan AWS Lambda.
- Menggunakan paket Permintaan untuk mengirim permintaan ke layanan web.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkap di GitHub.

Layanan yang digunakan dalam contoh ini

- API Gateway
- Aurora
- Lambda
- Manajer Rahasia

Untuk daftar lengkap AWS Panduan pengembang SDK dan contoh kode, lihat Menggunakan Lambda dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Membuat aplikasi messenger dengan Step Functions

Contoh kode berikut menunjukkan cara membuat AWS Step Functions aplikasi messenger yang mengambil catatan pesan dari tabel database.

Python

SDK untuk Python (Boto3)

Menunjukkan cara menggunakan AWS SDK untuk Python (Boto3) bersama AWS Step Functions untuk membuat aplikasi messenger yang mengambil catatan pesan dari tabel Amazon DynamoDB dan mengirimkannya dengan Amazon Simple Queue Service (Amazon SQS). Mesin negara terintegrasi dengan AWS Lambda fungsi untuk memindai database untuk pesan yang tidak terkirim.

- Buat mesin status yang mengambil dan memperbarui catatan pesan dari tabel Amazon DynamoDB.
- Perbarui definisi mesin status untuk mengirim pesan ke Amazon Simple Queue Service (Amazon SQS).
- Mulai dan hentikan berjalannya mesin status.
- Terhubung ke Lambda, DynamoDB, dan Amazon SQS dari mesin status menggunakan integrasi layanan.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkap di GitHub.

Layanan yang digunakan dalam contoh ini

- DynamoDB

- Lambda
- Amazon SQS
- Fungsi Langkah

Untuk daftar lengkap AWS Panduan pengembang SDK dan contoh kode, lihat Menggunakan Lambda dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Buat API rest dengan integrasi proxy fungsi

Contoh kode berikut ini menunjukkan cara untuk melakukan:

- Membuat peran IAM untuk eksekusi Lambda
- Membuat dan menerapkan fungsi Lambda
- Buat REST API
- Konfigurasikan integrasi proxy Lambda
- Menyebarkan dan menguji API
- Pembersihan sumber daya

Bash

AWS CLI dengan skrip Bash

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankan di Contoh tutorial pengembang repositori.

```
#!/bin/bash

set -euo pipefail

# Simple API Gateway Lambda Integration Script
# This script creates a REST API with Lambda proxy integration

# Generate random identifiers
```

```
FUNCTION_NAME="GetStartedLambdaProxyIntegration-$(openssl rand -hex 4)"
ROLE_NAME="GetStartedLambdaBasicExecutionRole-$(openssl rand -hex 4)"
API_NAME="LambdaProxyAPI-$(openssl rand -hex 4)"

# Get AWS account info
ACCOUNT_ID=$(aws sts get-caller-identity --query Account --output text)
REGION=$(aws configure get region || echo "us-east-1")

# Validate inputs
if [[ -z "$ACCOUNT_ID" ]] || [[ -z "$REGION" ]]; then
    echo "Error: Failed to retrieve AWS account information" >&2
    exit 1
fi

echo "Creating Lambda function code..."

# Create Lambda function code with input validation
cat > lambda_function.py << 'EOF'
import json
import logging

logger = logging.getLogger()
logger.setLevel(logging.INFO)

def lambda_handler(event, context):
    try:
        logger.info("Received event: %s", json.dumps(event))

        greeter = 'World'

        # Safely retrieve greeter from query string parameters
        query_params = event.get('queryStringParameters') or {}
        if isinstance(query_params, dict) and 'greeter' in query_params:
            greeter_value = query_params.get('greeter')
            if isinstance(greeter_value, str) and greeter_value:
                greeter = greeter_value

        # Safely retrieve greeter from multi-value headers
        multi_headers = event.get('multiValueHeaders') or {}
        if isinstance(multi_headers, dict) and 'greeter' in multi_headers:
            greeter_list = multi_headers.get('greeter', [])
            if isinstance(greeter_list, list) and greeter_list:
                greeter = " and ".join(str(g) for g in greeter_list if g)
```

```
        # Safely retrieve greeter from headers
        headers = event.get('headers') or {}
        if isinstance(headers, dict) and 'greeter' in headers:
            greeter_value = headers.get('greeter')
            if isinstance(greeter_value, str) and greeter_value:
                greeter = greeter_value

        # Safely retrieve greeter from body
        body = event.get('body')
        if body and isinstance(body, str):
            try:
                body_dict = json.loads(body)
                if isinstance(body_dict, dict) and 'greeter' in body_dict:
                    greeter_value = body_dict.get('greeter')
                    if isinstance(greeter_value, str) and greeter_value:
                        greeter = greeter_value
            except (json.JSONDecodeError, ValueError) as e:
                logger.warning("Failed to parse body: %s", str(e))

        # Sanitize greeter to prevent injection
        greeter = greeter.replace('"', '\\"').replace("'", "\\'")

        response = {
            "statusCode": 200,
            "headers": {
                "Content-Type": "application/json"
            },
            "body": json.dumps({"message": f"Hello, {greeter}!"})
        }

        logger.info("Response: %s", json.dumps(response))
        return response

    except Exception as e:
        logger.error("Unexpected error: %s", str(e), exc_info=True)
        return {
            "statusCode": 500,
            "headers": {
                "Content-Type": "application/json"
            },
            "body": json.dumps({"error": "Internal server error"})
        }
EOF
```

```
# Create deployment package
zip -q function.zip lambda_function.py || {
    echo "Error: Failed to create function.zip" >&2
    exit 1
}

echo "Creating IAM role..."

# Create IAM trust policy
cat > trust-policy.json << 'EOF'
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "lambda.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
EOF

# Create IAM role with error handling
aws iam create-role \
    --role-name "$ROLE_NAME" \
    --assume-role-policy-document file://trust-policy.json \
    --description "Temporary role for Lambda execution" || {
    echo "Error: Failed to create IAM role" >&2
    exit 1
}

aws iam tag-role --role-name "$ROLE_NAME" --tags Key=project,Value=doc-smith
 Key=tutorial,Value=apigateway-lambda-integration

# Attach execution policy
aws iam attach-role-policy \
    --role-name "$ROLE_NAME" \
    --policy-arn "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole" || {
    echo "Error: Failed to attach IAM policy" >&2
    exit 1
}
```

```
# Wait for role propagation
sleep 15

echo "Creating Lambda function..."

# Create Lambda function with Python 3.11 (more recent runtime)
aws lambda create-function \
    --function-name "$FUNCTION_NAME" \
    --runtime python3.11 \
    --role "arn:aws:iam::$ACCOUNT_ID:role/$ROLE_NAME" \
    --handler lambda_function.lambda_handler \
    --zip-file fileb://function.zip \
    --timeout 30 \
    --memory-size 128 \
    --environment "Variables={LOG_LEVEL=INFO}" \
    --tags project=doc-smith,tutorial=apigateway-lambda-integration || {
    echo "Error: Failed to create Lambda function" >&2
    exit 1
}

echo "Creating API Gateway..."

# Create REST API with minimum logging
API_RESPONSE=$(aws apigateway create-rest-api \
    --name "$API_NAME" \
    --endpoint-configuration types=REGIONAL \
    --description "API for Lambda proxy integration tutorial" \
    --tags project=doc-smith,tutorial=apigateway-lambda-integration \
    --output json)

API_ID=$(echo "$API_RESPONSE" | grep -o '"id": "[^"]*"' | head -1 | cut -d'"' -
f4)

if [[ -z "$API_ID" ]]; then
    echo "Error: Failed to create API Gateway" >&2
    exit 1
fi

# Get root resource ID
ROOT_RESOURCE_ID=$(aws apigateway get-resources --rest-api-id "$API_ID" --query
 'items[?path==`/`].id' --output text)

# Create helloworld resource
```

```
aws apigateway create-resource \
    --rest-api-id "$API_ID" \
    --parent-id "$ROOT_RESOURCE_ID" \
    --path-part helloworld || {
    echo "Error: Failed to create resource" >&2
    exit 1
}

# Get resource ID
RESOURCE_ID=$(aws apigateway get-resources --rest-api-id "$API_ID" --query
 "items[?pathPart=='helloworld'].id" --output text)

# Create ANY method with no authorization (intentional for tutorial)
aws apigateway put-method \
    --rest-api-id "$API_ID" \
    --resource-id "$RESOURCE_ID" \
    --http-method ANY \
    --authorization-type NONE || {
    echo "Error: Failed to create method" >&2
    exit 1
}

# Set up Lambda proxy integration
LAMBDA_URI="arn:aws:apigateway:$REGION:lambda:path/2015-03-31/functions/
arn:aws:lambda:$REGION:$ACCOUNT_ID:function:$FUNCTION_NAME/invocations"

aws apigateway put-integration \
    --rest-api-id "$API_ID" \
    --resource-id "$RESOURCE_ID" \
    --http-method ANY \
    --type AWS_PROXY \
    --integration-http-method POST \
    --uri "$LAMBDA_URI" || {
    echo "Error: Failed to create integration" >&2
    exit 1
}

# Grant API Gateway permission to invoke Lambda
STATEMENT_ID="apigateway-invoke-$(openssl rand -hex 4)"
SOURCE_ARN="arn:aws:execute-api:$REGION:$ACCOUNT_ID:$API_ID/*/*"

aws lambda add-permission \
    --function-name "$FUNCTION_NAME" \
    --statement-id "$STATEMENT_ID" \
```

```
    --action lambda:InvokeFunction \
    --principal apigateway.amazonaws.com \
    --source-arn "$SOURCE_ARN" || {
    echo "Error: Failed to add Lambda permission" >&2
    exit 1
}

# Deploy API
aws apigateway create-deployment \
    --rest-api-id "$API_ID" \
    --stage-name test \
    --description "Test deployment" || {
    echo "Error: Failed to deploy API" >&2
    exit 1
}

echo "Testing API..."

# Test the API
INVOKE_URL="https://$API_ID.execute-api.$REGION.amazonaws.com/test/helloworld"

echo "API URL: $INVOKE_URL"

# Test with query parameter (with proper URL encoding)
echo "Testing with query parameter:"
curl -s -X GET "$INVOKE_URL?greeter=John" | jq . 2>/dev/null || curl -s -X GET
 "$INVOKE_URL?greeter=John"
echo ""

# Test with header
echo "Testing with header:"
curl -s -X GET "$INVOKE_URL" \
    -H 'content-type: application/json' \
    -H 'greeter: John' | jq . 2>/dev/null || curl -s -X GET "$INVOKE_URL" \
    -H 'content-type: application/json' \
    -H 'greeter: John'
echo ""

# Test with body
echo "Testing with POST body:"
curl -s -X POST "$INVOKE_URL" \
    -H 'content-type: application/json' \
    -d '{"greeter": "John"}' | jq . 2>/dev/null || curl -s -X POST "$INVOKE_URL"
 \
```

```
    -H 'content-type: application/json' \
    -d '{"greeter": "John"}'
echo ""

echo "Tutorial completed! API is available at: $INVOKE_URL"

# Cleanup
echo "Cleaning up resources..."

# Delete API
aws apigateway delete-rest-api --rest-api-id "$API_ID" || echo "Warning: Failed
 to delete API" >&2

# Delete Lambda function
aws lambda delete-function --function-name "$FUNCTION_NAME" || echo "Warning:
 Failed to delete Lambda function" >&2

# Detach policy and delete role
aws iam detach-role-policy \
    --role-name "$ROLE_NAME" \
    --policy-arn "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole" || echo "Warning: Failed to detach policy" >&2

aws iam delete-role --role-name "$ROLE_NAME" || echo "Warning: Failed to delete
 role" >&2

# Clean up local files securely
rm -f lambda_function.py function.zip trust-policy.json

echo "Cleanup completed!"
```

- Untuk detail API, lihat topik berikut di Referensi Perintah AWS CLI .
  - AddPermission
  - AttachRolePolicy
  - CreateDeployment
  - CreateFunction
  - CreateResource
  - CreateRestApi
  - CreateRole
  - DeleteFunction

- DeleteRestApi
- DeleteRole
- DetachRolePolicy
- GetCallerIdentity
- GetResources
- GetRestApis
- PutIntegration
- PutMethod

Untuk daftar lengkap AWS Panduan pengembang SDK dan contoh kode, lihat Menggunakan Lambda dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Membuat aplikasi manajemen aset foto yang memungkinkan pengguna mengelola foto menggunakan label

Contoh kode berikut ini menunjukkan cara membuat aplikasi nirserver yang memungkinkan pengguna mengelola foto menggunakan label.

.NET

SDK untuk .NET

Menunjukkan cara mengembangkan aplikasi manajemen aset foto yang mendeteksi label dalam gambar menggunakan Amazon Rekognition dan menyimpannya untuk pengambilan nanti.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkap di GitHub.

Untuk mendalami tentang asal usul contoh ini, lihat postingan di Komunitas AWS.

Layanan yang digunakan dalam contoh ini

- API Gateway
- DynamoDB
- Lambda

- Amazon Rekognition
- Amazon S3
- Amazon SNS

C++

SDK untuk C++

Menunjukkan cara mengembangkan aplikasi manajemen aset foto yang mendeteksi label dalam gambar menggunakan Amazon Rekognition dan menyimpannya untuk pengambilan nanti.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkap di GitHub.

Untuk mendalami tentang asal usul contoh ini, lihat postingan di Komunitas AWS.

Layanan yang digunakan dalam contoh ini

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Java

SDK untuk Java 2.x

Menunjukkan cara mengembangkan aplikasi manajemen aset foto yang mendeteksi label dalam gambar menggunakan Amazon Rekognition dan menyimpannya untuk pengambilan nanti.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkap di GitHub.

Untuk mendalami tentang asal usul contoh ini, lihat postingan di Komunitas AWS.

Layanan yang digunakan dalam contoh ini

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

JavaScript

SDK untuk JavaScript (V3)

Menunjukkan cara mengembangkan aplikasi manajemen aset foto yang mendeteksi label dalam gambar menggunakan Amazon Rekognition dan menyimpannya untuk pengambilan nanti.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkap di GitHub.

Untuk mendalami tentang asal usul contoh ini, lihat postingan di Komunitas AWS.

Layanan yang digunakan dalam contoh ini

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Kotlin

SDK untuk Kotlin

Menunjukkan cara mengembangkan aplikasi manajemen aset foto yang mendeteksi label dalam gambar menggunakan Amazon Rekognition dan menyimpannya untuk pengambilan nanti.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkap di GitHub.

Untuk mendalami tentang asal usul contoh ini, lihat postingan di Komunitas AWS.

Layanan yang digunakan dalam contoh ini

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

PHP

SDK untuk PHP

Menunjukkan cara mengembangkan aplikasi manajemen aset foto yang mendeteksi label dalam gambar menggunakan Amazon Rekognition dan menyimpannya untuk pengambilan nanti.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkap di GitHub.

Untuk mendalami tentang asal usul contoh ini, lihat postingan di Komunitas AWS.

Layanan yang digunakan dalam contoh ini

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Rust

SDK untuk Rust

Menunjukkan cara mengembangkan aplikasi manajemen aset foto yang mendeteksi label dalam gambar menggunakan Amazon Rekognition dan menyimpannya untuk pengambilan nanti.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkap di GitHub.

Untuk mendalami tentang asal usul contoh ini, lihat postingan di Komunitas AWS.

Layanan yang digunakan dalam contoh ini

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Untuk daftar lengkap AWS Panduan pengembang SDK dan contoh kode, lihat Menggunakan Lambda dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Membuat aplikasi obrolan websocket dengan API Gateway

Contoh kode berikut menunjukkan cara membuat aplikasi obrolan yang dilayani oleh API websocket yang dibangun di Amazon API Gateway.

Python

SDK untuk Python (Boto3)

Menunjukkan cara menggunakan AWS SDK untuk Python (Boto3) dengan Amazon API Gateway V2 untuk membuat API websocket yang terintegrasi dengan AWS Lambda dan Amazon DynamoDB.

- Buat API websocket yang dilayani oleh API Gateway.
- Tentukan penangan Lambda yang menyimpan koneksi di DynamoDB dan memposting pesan ke peserta obrolan lainnya.
- Hubungkan ke aplikasi obrolan websocket dan kirim pesan dengan paket Websocket.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkap di GitHub.

Layanan yang digunakan dalam contoh ini

- API Gateway
- DynamoDB
- Lambda

Untuk daftar lengkap AWS Panduan pengembang SDK dan contoh kode, lihat Menggunakan Lambda dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Buat aplikasi yang menganalisis umpan balik pelanggan dan mensintesis audio

Contoh kode berikut menunjukkan cara membuat aplikasi yang menganalisis kartu komentar pelanggan, menerjemahkannya dari bahasa aslinya, menentukan sentimen mereka, dan menghasilkan file audio dari teks yang diterjemahkan.

.NET

SDK untuk .NET

Aplikasi contoh ini menganalisis dan menyimpan kartu umpan balik pelanggan. Secara khusus, itu memenuhi kebutuhan hotel fiktif di New York City. Hotel menerima umpan balik dari para tamu dalam berbagai bahasa dalam bentuk kartu komentar fisik. Umpan balik itu diunggah ke aplikasi melalui klien web. Setelah gambar kartu komentar diunggah, langkah-langkah berikut terjadi:

- Teks diekstraksi dari gambar menggunakan Amazon Textract.
- Amazon Comprecept menentukan sentimen teks yang diekstraksi dan bahasanya.
- Teks yang diekstraksi diterjemahkan ke bahasa Inggris menggunakan Amazon Translate.

- Amazon Polly mensintesis file audio dari teks yang diekstrak.

Aplikasi lengkap dapat digunakan dengan AWS CDK. Untuk kode sumber dan instruksi penerapan, lihat proyek di GitHub.

Layanan yang digunakan dalam contoh ini

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

Java

SDK untuk Java 2.x

Aplikasi contoh ini menganalisis dan menyimpan kartu umpan balik pelanggan. Secara khusus, itu memenuhi kebutuhan hotel fiktif di New York City. Hotel menerima umpan balik dari para tamu dalam berbagai bahasa dalam bentuk kartu komentar fisik. Umpan balik itu diunggah ke aplikasi melalui klien web. Setelah gambar kartu komentar diunggah, langkah-langkah berikut terjadi:

- Teks diekstraksi dari gambar menggunakan Amazon Textract.
- Amazon Comprecept menentukan sentimen teks yang diekstraksi dan bahasanya.
- Teks yang diekstraksi diterjemahkan ke bahasa Inggris menggunakan Amazon Translate.
- Amazon Polly mensintesis file audio dari teks yang diekstrak.

Aplikasi lengkap dapat digunakan dengan AWS CDK. Untuk kode sumber dan instruksi penerapan, lihat proyek di GitHub.

Layanan yang digunakan dalam contoh ini

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

JavaScript

SDK untuk JavaScript (V3)

Aplikasi contoh ini menganalisis dan menyimpan kartu umpan balik pelanggan. Secara khusus, itu memenuhi kebutuhan hotel fiktif di New York City. Hotel menerima umpan balik dari para tamu dalam berbagai bahasa dalam bentuk kartu komentar fisik. Umpan balik itu diunggah ke aplikasi melalui klien web. Setelah gambar kartu komentar diunggah, langkah-langkah berikut terjadi:

- Teks diekstraksi dari gambar menggunakan Amazon Textract.
- Amazon Comprecept menentukan sentimen teks yang diekstraksi dan bahasanya.
- Teks yang diekstraksi diterjemahkan ke bahasa Inggris menggunakan Amazon Translate.
- Amazon Polly mensintesis file audio dari teks yang diekstrak.

Aplikasi lengkap dapat digunakan dengan AWS CDK. Untuk kode sumber dan instruksi penerapan, lihat proyek di GitHub. Kutipan berikut menunjukkan bagaimana AWS SDK untuk JavaScript digunakan di dalam fungsi Lambda.

```
import {
  ComprehendClient,
  DetectDominantLanguageCommand,
  DetectSentimentCommand,
} from "@aws-sdk/client-comprehend";

/**
 * Determine the language and sentiment of the extracted text.
 *
 * @param {{ source_text: string}} extractTextOutput
 */
export const handler = async (extractTextOutput) => {
  const comprehendClient = new ComprehendClient({});

  const detectDominantLanguageCommand = new DetectDominantLanguageCommand({
    Text: extractTextOutput.source_text,
  });

  // The source language is required for sentiment analysis and
  // translation in the next step.
  const { Languages } = await comprehendClient.send(
    detectDominantLanguageCommand,
  );
```

```
  const languageCode = Languages[0].LanguageCode;

  const detectSentimentCommand = new DetectSentimentCommand({
    Text: extractTextOutput.source_text,
    LanguageCode: languageCode,
  });

  const { Sentiment } = await comprehendClient.send(detectSentimentCommand);

  return {
    sentiment: Sentiment,
    language_code: languageCode,
  };
};
```

```
import {
  DetectDocumentTextCommand,
  TextractClient,
} from "@aws-sdk/client-textract";

/**
 * Fetch the S3 object from the event and analyze it using Amazon Textract.
 *
 * @param {import("@types/aws-lambda").EventBridgeEvent<"Object Created">}
 eventBridgeS3Event
 */
export const handler = async (eventBridgeS3Event) => {
  const textractClient = new TextractClient();

  const detectDocumentTextCommand = new DetectDocumentTextCommand({
    Document: {
      S3Object: {
        Bucket: eventBridgeS3Event.bucket,
        Name: eventBridgeS3Event.object,
      },
    },
  });

  // Textract returns a list of blocks. A block can be a line, a page, word, etc.
  // Each block also contains geometry of the detected text.
  // For more information on the Block type, see https://docs.aws.amazon.com/
textract/latest/dg/API_Block.html.
```

```
  const { Blocks } = await textractClient.send(detectDocumentTextCommand);

  // For the purpose of this example, we are only interested in words.
  const extractedWords = Blocks.filter((b) => b.BlockType === "WORD").map(
    (b) => b.Text,
  );

  return extractedWords.join(" ");
};
```

```
import { PollyClient, SynthesizeSpeechCommand } from "@aws-sdk/client-polly";
import { S3Client } from "@aws-sdk/client-s3";
import { Upload } from "@aws-sdk/lib-storage";

/**
 * Synthesize an audio file from text.
 *
 * @param {{ bucket: string, translated_text: string, object: string}}
 sourceDestinationConfig
 */
export const handler = async (sourceDestinationConfig) => {
  const pollyClient = new PollyClient({});

  const synthesizeSpeechCommand = new SynthesizeSpeechCommand({
    Engine: "neural",
    Text: sourceDestinationConfig.translated_text,
    VoiceId: "Ruth",
    OutputFormat: "mp3",
  });

  const { AudioStream } = await pollyClient.send(synthesizeSpeechCommand);

  const audioKey = `${sourceDestinationConfig.object}.mp3`;

  // Store the audio file in S3.
  const s3Client = new S3Client();
  const upload = new Upload({
    client: s3Client,
    params: {
      Bucket: sourceDestinationConfig.bucket,
      Key: audioKey,
      Body: AudioStream,
      ContentType: "audio/mp3",
```

```
    },
  });

  await upload.done();
  return audioKey;
};
```

```
import {
  TranslateClient,
  TranslateTextCommand,
} from "@aws-sdk/client-translate";

/**
 * Translate the extracted text to English.
 *
 * @param {{ extracted_text: string, source_language_code: string}}
 textAndSourceLanguage
 */
export const handler = async (textAndSourceLanguage) => {
  const translateClient = new TranslateClient({});

  const translateCommand = new TranslateTextCommand({
    SourceLanguageCode: textAndSourceLanguage.source_language_code,
    TargetLanguageCode: "en",
    Text: textAndSourceLanguage.extracted_text,
  });

  const { TranslatedText } = await translateClient.send(translateCommand);

  return { translated_text: TranslatedText };
};
```

Layanan yang digunakan dalam contoh ini

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

Ruby

SDK untuk Ruby

Aplikasi contoh ini menganalisis dan menyimpan kartu umpan balik pelanggan. Secara khusus, itu memenuhi kebutuhan hotel fiktif di New York City. Hotel menerima umpan balik dari para tamu dalam berbagai bahasa dalam bentuk kartu komentar fisik. Umpan balik itu diunggah ke aplikasi melalui klien web. Setelah gambar kartu komentar diunggah, langkah-langkah berikut terjadi:

- Teks diekstraksi dari gambar menggunakan Amazon Textract.
- Amazon Comprecept menentukan sentimen teks yang diekstraksi dan bahasanya.
- Teks yang diekstraksi diterjemahkan ke bahasa Inggris menggunakan Amazon Translate.
- Amazon Polly mensintesis file audio dari teks yang diekstrak.

Aplikasi lengkap dapat digunakan dengan AWS CDK. Untuk kode sumber dan instruksi penerapan, lihat proyek di GitHub.

Layanan yang digunakan dalam contoh ini

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

Untuk daftar lengkap AWS Panduan pengembang SDK dan contoh kode, lihat Menggunakan Lambda dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Membuat dasbor pemantauan dengan nama fungsi sebagai variabel

Contoh kode berikut ini menunjukkan cara untuk melakukan:

- Buat CloudWatch dasbor
- Tambahkan widget metrik Lambda dengan variabel nama fungsi
- Verifikasi dasbor
- Pembersihan sumber daya

Bash

AWS CLI dengan skrip Bash

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankan di Contoh tutorial pengembang repositori.

```
#!/bin/bash

# Script to create a CloudWatch dashboard with Lambda function name as a variable
# This script creates a CloudWatch dashboard that allows you to switch between
 different Lambda functions

# Set up logging with secure permissions
LOG_FILE="${HOME}/.cloudwatch-dashboard-script.log"
touch "$LOG_FILE" && chmod 600 "$LOG_FILE"
exec > >(tee -a "$LOG_FILE") 2>&1

echo "$(date): Starting CloudWatch dashboard creation script"

# Security: Set strict error handling
set -uo pipefail
trap 'handle_error "Script failed at line $LINENO"' ERR

# Function to handle errors
handle_error() {
    local error_msg="${1:-Unknown error}"
    echo "ERROR: $error_msg" >&2
    echo "Resources created:"
    echo "- CloudWatch Dashboard: LambdaMetricsDashboard"
    echo ""
    echo "========================================="
    echo "CLEANUP CONFIRMATION"
    echo "========================================="
    echo "An error occurred. Proceeding with automatic cleanup..."

    echo "Cleaning up resources..."
    aws cloudwatch delete-dashboards --dashboard-names LambdaMetricsDashboard 2>/
dev/null || true
```

```
    # Clean up temporary files
    if [ -n "${TEMP_DIR:-}" ] && [ -d "$TEMP_DIR" ]; then
        rm -rf "$TEMP_DIR"
    fi
    rm -f dashboard-body.json

    echo "Cleanup complete."
    exit 1
}

# Security: Validate AWS CLI is installed
if ! command -v aws &> /dev/null; then
    handle_error "AWS CLI is not installed. Please install it and try again."
fi

# Check if AWS CLI is installed and configured
echo "Checking AWS CLI configuration..."
if ! aws sts get-caller-identity > /dev/null 2>&1; then
    handle_error "AWS CLI is not properly configured. Please configure it with
 'aws configure' and try again."
fi

# Get the current region securely
REGION=$(aws configure get region 2>/dev/null || echo "")
if [ -z "$REGION" ]; then
    REGION="us-east-1"
    echo "No region found in AWS config, defaulting to $REGION"
fi
echo "Using region: $REGION"

# Validate region format
if ! [[ "$REGION" =~ ^[a-z]{2}-[a-z]+-[0-9]{1}$ ]]; then
    handle_error "Invalid AWS region format: $REGION"
fi

# Check if there are any Lambda functions in the account
echo "Checking for Lambda functions..."
LAMBDA_FUNCTIONS=$(aws lambda list-functions --region "$REGION" --query
 "Functions[*].FunctionName" --output text 2>/dev/null || echo "")

if [ -z "$LAMBDA_FUNCTIONS" ]; then
    echo "No Lambda functions found in your account. Creating a simple test
 function..."
```

```
    # Create a temporary directory for Lambda function code with secure
 permissions
    TEMP_DIR=$(mktemp -d)
    chmod 700 "$TEMP_DIR"
    trap 'rm -rf "$TEMP_DIR"' EXIT

    # Create a simple Lambda function
    cat > "$TEMP_DIR/index.js" << 'EOF'
exports.handler = async (event) => {
    console.log('Event:', JSON.stringify(event, null, 2));
    return {
        statusCode: 200,
        body: JSON.stringify('Hello from Lambda!'),
    };
};
EOF

    # Zip the function code
    if ! cd "$TEMP_DIR"; then
        handle_error "Failed to change to temporary directory"
    fi

    if ! zip -q function.zip index.js; then
        handle_error "Failed to create zip file"
    fi

    # Create a role for the Lambda function with restricted trust policy
    ROLE_NAME="LambdaDashboardTestRole-$(date +%s)"
    TRUST_POLICY='{"Version":"2012-10-17",       "Statement":
[{"Effect":"Allow","Principal":
{"Service":"lambda.amazonaws.com"},"Action":"sts:AssumeRole"}]}'

    if ! ROLE_ARN=$(aws iam create-role \
        --role-name "$ROLE_NAME" \
        --assume-role-policy-document "$TRUST_POLICY" \
        --tags Key=project,Value=doc-smith Key=tutorial,Value=cloudwatch-
dynamicdash \
        --query "Role.Arn" \
        --output text 2>/dev/null); then
        handle_error "Failed to create IAM role for Lambda function"
    fi

    echo "Waiting for role to be available..."
```

```
    sleep 10

    # Attach basic Lambda execution policy
    if ! aws iam attach-role-policy \
        --role-name "$ROLE_NAME" \
        --policy-arn "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole"; then
        aws iam delete-role --role-name "$ROLE_NAME" 2>/dev/null || true
        handle_error "Failed to attach policy to IAM role"
    fi

    # Create the Lambda function
    FUNCTION_NAME="DashboardTestFunction-$(date +%s)"
    if ! aws lambda create-function \
        --function-name "$FUNCTION_NAME" \
        --runtime nodejs18.x \
        --role "$ROLE_ARN" \
        --handler index.handler \
        --zip-file fileb://function.zip \
        --tags project=doc-smith,tutorial=cloudwatch-dynamicdash \
        --region "$REGION" > /dev/null 2>&1; then
        aws iam detach-role-policy \
            --role-name "$ROLE_NAME" \
            --policy-arn "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole" 2>/dev/null || true
        aws iam delete-role --role-name "$ROLE_NAME" 2>/dev/null || true
        handle_error "Failed to create Lambda function"
    fi

    # Invoke the function to generate some metrics
    echo "Invoking Lambda function to generate metrics..."
    for i in {1..5}; do
        aws lambda invoke --function-name "$FUNCTION_NAME" --payload '{}' /dev/
null --region "$REGION" > /dev/null 2>&1 || true
        sleep 1
    done

    # Go back to original directory
    cd - > /dev/null

    # Set the function name for the dashboard
    DEFAULT_FUNCTION="$FUNCTION_NAME"
else
    # Use the first Lambda function as default
```

```
    DEFAULT_FUNCTION=$(echo "$LAMBDA_FUNCTIONS" | awk '{print $1}')
    echo "Found Lambda functions. Using $DEFAULT_FUNCTION as default."
    FUNCTION_NAME=""
    ROLE_NAME=""
fi

# Create a dashboard with Lambda metrics and a function name variable
echo "Creating CloudWatch dashboard with Lambda function name variable..."

# Create a JSON file for the dashboard body with secure permissions
DASHBOARD_JSON="dashboard-body-$$.json"
touch "$DASHBOARD_JSON" && chmod 600 "$DASHBOARD_JSON"

# Escape special characters in region and function name for JSON
REGION_ESCAPED=$(printf '%s\n' "$REGION" | sed 's:[\/&]:\\&:g')
FUNCTION_ESCAPED=$(printf '%s\n' "$DEFAULT_FUNCTION" | sed 's:[\/&]:\\&:g')

cat > "$DASHBOARD_JSON" << EOF
{
  "widgets": [
    {
      "type": "metric",
      "x": 0,
      "y": 0,
      "width": 12,
      "height": 6,
      "properties": {
        "metrics": [
          [ "AWS/Lambda", "Invocations", "FunctionName", "\${FunctionName}" ],
          [ ".", "Errors", ".", "." ],
          [ ".", "Throttles", ".", "." ]
        ],
        "view": "timeSeries",
        "stacked": false,
        "region": "$REGION_ESCAPED",
        "title": "Lambda Function Metrics for \${FunctionName}",
        "period": 300
      }
    },
    {
      "type": "metric",
      "x": 0,
      "y": 6,
      "width": 12,
```

```
      "height": 6,
      "properties": {
        "metrics": [
          [ "AWS/Lambda", "Duration", "FunctionName", "\${FunctionName}",
 { "stat": "Average" } ]
        ],
        "view": "timeSeries",
        "stacked": false,
        "region": "$REGION_ESCAPED",
        "title": "Duration for \${FunctionName}",
        "period": 300
      }
    },
    {
      "type": "metric",
      "x": 12,
      "y": 0,
      "width": 12,
      "height": 6,
      "properties": {
        "metrics": [
          [ "AWS/Lambda", "ConcurrentExecutions", "FunctionName", "\
${FunctionName}" ]
        ],
        "view": "timeSeries",
        "stacked": false,
        "region": "$REGION_ESCAPED",
        "title": "Concurrent Executions for \${FunctionName}",
        "period": 300
      }
    }
  ],
  "periodOverride": "auto",
  "variables": [
    {
      "type": "property",
      "id": "FunctionName",
      "property": "FunctionName",
      "label": "Lambda Function",
      "inputType": "select",
      "values": [
        {
          "value": "$FUNCTION_ESCAPED",
          "label": "$FUNCTION_ESCAPED"
```

```
        }
      ]
    }
  ]
}
EOF

# Validate JSON before sending
if ! jq empty "$DASHBOARD_JSON" 2>/dev/null; then
    handle_error "Invalid JSON generated for dashboard"
fi

# Create the dashboard using the JSON file
DASHBOARD_NAME="LambdaMetricsDashboard-$(date +%s)"
if ! DASHBOARD_RESULT=$(aws cloudwatch put-dashboard \
    --dashboard-name "$DASHBOARD_NAME" \
    --dashboard-body file://"$DASHBOARD_JSON" \
    --region "$REGION" 2>&1); then
    # If we created resources, clean them up
    if [ -n "${FUNCTION_NAME:-}" ]; then
        aws lambda delete-function --function-name "$FUNCTION_NAME" --region
 "$REGION" 2>/dev/null || true
        aws iam detach-role-policy \
            --role-name "$ROLE_NAME" \
            --policy-arn "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole" 2>/dev/null || true
        aws iam delete-role --role-name "$ROLE_NAME" 2>/dev/null || true
    fi
    handle_error "Failed to create CloudWatch dashboard."
fi

# Display any validation messages but continue
if echo "$DASHBOARD_RESULT" | grep -q "DashboardValidationMessages"; then
    echo "Dashboard created with validation messages:"
    echo "$DASHBOARD_RESULT"
    echo "These validation messages are warnings and the dashboard should still
 function."
else
    echo "Dashboard created successfully!"
fi

# Verify the dashboard was created
echo "Verifying dashboard creation..."
```

```
if ! DASHBOARD_INFO=$(aws cloudwatch get-dashboard --dashboard-name
 "$DASHBOARD_NAME" --region "$REGION" 2>&1); then
    # If we created resources, clean them up
    if [ -n "${FUNCTION_NAME:-}" ]; then
        aws lambda delete-function --function-name "$FUNCTION_NAME" --region
 "$REGION" 2>/dev/null || true
        aws iam detach-role-policy \
            --role-name "$ROLE_NAME" \
            --policy-arn "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole" 2>/dev/null || true
        aws iam delete-role --role-name "$ROLE_NAME" 2>/dev/null || true
    fi
    handle_error "Failed to verify dashboard creation."
fi

echo "Dashboard verification successful!"
echo "Dashboard details:"
echo "$DASHBOARD_INFO"

# List all dashboards to confirm
echo "Listing all dashboards:"
if ! DASHBOARDS=$(aws cloudwatch list-dashboards --region "$REGION" 2>&1); then
    # If we created resources, clean them up
    if [ -n "${FUNCTION_NAME:-}" ]; then
        aws lambda delete-function --function-name "$FUNCTION_NAME" --region
 "$REGION" 2>/dev/null || true
        aws iam detach-role-policy \
            --role-name "$ROLE_NAME" \
            --policy-arn "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole" 2>/dev/null || true
        aws iam delete-role --role-name "$ROLE_NAME" 2>/dev/null || true
    fi
    handle_error "Failed to list dashboards."
fi
echo "$DASHBOARDS"

# Show instructions for accessing the dashboard
echo ""
echo "Dashboard created successfully! To access it:"
echo "1. Open the CloudWatch console at https://console.aws.amazon.com/
cloudwatch/"
echo "2. In the navigation pane, choose Dashboards"
echo "3. Select $DASHBOARD_NAME"
```

```
echo "4. You should see a dropdown menu labeled 'Lambda Function' at the top of
 the dashboard"
echo "5. Use this dropdown to select different Lambda functions and see their
 metrics"
echo ""

# Create a list of resources for cleanup
RESOURCES=("- CloudWatch Dashboard: $DASHBOARD_NAME")
if [ -n "${FUNCTION_NAME:-}" ]; then
    RESOURCES+=("- Lambda Function: $FUNCTION_NAME")
    RESOURCES+=("- IAM Role: $ROLE_NAME")
fi

# Prompt for cleanup with automatic yes
echo "========================================="
echo "CLEANUP CONFIRMATION"
echo "========================================="
echo "Resources created:"
for resource in "${RESOURCES[@]}"; do
    echo "$resource"
done
echo ""
echo "Proceeding with automatic cleanup..."

CLEANUP_CHOICE="y"

if [[ "${CLEANUP_CHOICE,,}" == "y" ]]; then
    echo "Cleaning up resources..."

    # Delete the dashboard
    if aws cloudwatch delete-dashboards --dashboard-names "$DASHBOARD_NAME" --
region "$REGION" 2>/dev/null; then
        echo "Dashboard deleted successfully."
    else
        echo "WARNING: Failed to delete dashboard. You may need to delete it
 manually."
    fi

    # If we created a Lambda function, delete it and its role
    if [ -n "${FUNCTION_NAME:-}" ]; then
        echo "Deleting Lambda function..."
        if aws lambda delete-function --function-name "$FUNCTION_NAME" --region
 "$REGION" 2>/dev/null; then
            echo "Lambda function deleted successfully."
```

```
        else
            echo "WARNING: Failed to delete Lambda function. You may need to
 delete it manually."
        fi

        echo "Detaching role policy..."
        if aws iam detach-role-policy \
            --role-name "$ROLE_NAME" \
            --policy-arn "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole" 2>/dev/null; then
            echo "Role policy detached successfully."
        else
            echo "WARNING: Failed to detach role policy. You may need to detach
 it manually."
        fi

        echo "Deleting IAM role..."
        if aws iam delete-role --role-name "$ROLE_NAME" 2>/dev/null; then
            echo "IAM role deleted successfully."
        else
            echo "WARNING: Failed to delete IAM role. You may need to delete it
 manually."
        fi
    fi

    # Clean up the JSON file
    rm -f "$DASHBOARD_JSON"

    echo "Cleanup complete."
fi

echo "Script completed successfully!"
```

- Untuk detail API, lihat topik berikut di Referensi Perintah AWS CLI .
  - AttachRolePolicy
  - CreateFunction
  - CreateRole
  - DeleteDashboards
  - DeleteFunction
  - DeleteRole

- DetachRolePolicy
- GetCallerIdentity
- GetDashboard
- Memanggil
- ListDashboards
- ListFunctions
- PutDashboard

Untuk daftar lengkap AWS Panduan pengembang SDK dan contoh kode, lihat Menggunakan Lambda dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Membuat fungsi tanpa server pertama Anda

Contoh kode berikut ini menunjukkan cara untuk melakukan:

- Membuat peran IAM untuk Lambda
- Buat kode fungsi
- Buat fungsi Lambda
- Uji fungsi Lambda Anda
- Pembersihan sumber daya

Bash

AWS CLI dengan skrip Bash

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankan di Contoh tutorial pengembang repositori.

```
#!/bin/bash
# AWS Lambda - Create Your First Function
# This script creates a Lambda function, invokes it with a test event,
# views CloudWatch logs, and cleans up all resources.
```

```
#
# Source: https://docs.aws.amazon.com/lambda/latest/dg/getting-started.html
#
# Resources created:
#   - IAM role (Lambda execution role with basic logging permissions)
#   - Lambda function (Python 3.13 or Node.js 22.x runtime)
#   - CloudWatch log group (created automatically by Lambda on invocation)

set -eE -o pipefail

################################################################################
# Setup
################################################################################

UNIQUE_ID=$(head -c 8 /dev/urandom | od -An -tx1 | tr -d ' ')
FUNCTION_NAME="my-lambda-function-${UNIQUE_ID}"
ROLE_NAME="lambda-execution-role-${UNIQUE_ID}"
LOG_GROUP_NAME="/aws/lambda/${FUNCTION_NAME}"

TEMP_DIR=$(mktemp -d)
readonly TEMP_DIR
LOG_FILE="${TEMP_DIR}/lambda-gettingstarted.log"

exec > >(tee -a "$LOG_FILE") 2>&1

declare -a CREATED_RESOURCES

################################################################################
# Helper functions
################################################################################

cleanup_resources() {
    # Disable error trap to prevent recursion during cleanup
    trap - ERR
    set +eE

    echo ""
    echo "Cleaning up resources..."
    echo ""

    for ((i=${#CREATED_RESOURCES[@]}-1; i>=0; i--)); do
        local RESOURCE="${CREATED_RESOURCES[$i]}"
        local TYPE="${RESOURCE%%:*}"
        local NAME="${RESOURCE#*:}"
```

```
        case "$TYPE" in
            log-group)
                echo "Deleting CloudWatch log group: ${NAME}"
                aws logs delete-log-group \
                    --log-group-name "$NAME" 2>&1 || echo "  WARNING: Could not
delete log group ${NAME}."
                ;;
            lambda-function)
                echo "Deleting Lambda function: ${NAME}"
                aws lambda delete-function \
                    --function-name "$NAME" 2>&1 || echo "  WARNING: Could not
delete Lambda function ${NAME}."
                echo "  Waiting for function deletion to complete..."
                local DELETE_WAIT=0
                while aws lambda get-function --function-name "$NAME" > /dev/null
2>&1; do
                    sleep 2
                    DELETE_WAIT=$((DELETE_WAIT + 2))
                    if [ "$DELETE_WAIT" -ge 60 ]; then
                        echo "  WARNING: Timed out waiting for function
deletion."
                        break
                    fi
                done
                ;;
            iam-role-policy)
                local ROLE_PART="${NAME%%|*}"
                local POLICY_PART="${NAME#*|}"
                echo "Detaching policy from role: ${ROLE_PART}"
                aws iam detach-role-policy \
                    --role-name "$ROLE_PART" \
                    --policy-arn "$POLICY_PART" 2>&1 || echo "  WARNING: Could
not detach policy from role ${ROLE_PART}."
                ;;
            iam-role)
                echo "Deleting IAM role: ${NAME}"
                aws iam delete-role \
                    --role-name "$NAME" 2>&1 || echo "  WARNING: Could not delete
IAM role ${NAME}."
                ;;
        esac
    done
```

```
    if [ -d "$TEMP_DIR" ]; then
        rm -rf "$TEMP_DIR"
    fi

    echo ""
    echo "Cleanup complete."
}

handle_error() {
    echo ""
    echo "========================================="
    echo "ERROR: Script failed at $1"
    echo "========================================="
    echo ""
    if [ ${#CREATED_RESOURCES[@]} -gt 0 ]; then
        echo "Attempting to clean up ${#CREATED_RESOURCES[@]} resource(s)..."
        cleanup_resources
    fi
    exit 1
}

trap 'handle_error "line $LINENO"' ERR

wait_for_resource() {
    local DESCRIPTION="$1"
    local COMMAND="$2"
    local TARGET_VALUE="$3"
    local TIMEOUT=300
    local ELAPSED=0
    local INTERVAL=5

    echo "Waiting for ${DESCRIPTION}..."
    while true; do
        local RESULT
        RESULT=$(eval "$COMMAND" 2>&1) || true
        if echo "$RESULT" | grep -q "$TARGET_VALUE"; then
            echo "  ${DESCRIPTION} is ready."
            return 0
        fi
        if [ "$ELAPSED" -ge "$TIMEOUT" ]; then
            echo "ERROR: Timed out waiting for ${DESCRIPTION} after ${TIMEOUT}
 seconds."
            return 1
        fi
```

```
        sleep "$INTERVAL"
        ELAPSED=$((ELAPSED + INTERVAL))
    done
}

validate_input() {
    local input="$1"
    local pattern="$2"
    if ! [[ "$input" =~ $pattern ]]; then
        echo "ERROR: Invalid input: $input"
        return 1
    fi
    return 0
}

# ============================================================================
# Region pre-check
# ============================================================================

CONFIGURED_REGION=$(aws configure get region 2>/dev/null || true)
if [ -z "$CONFIGURED_REGION" ] && [ -z "$AWS_DEFAULT_REGION" ] && [ -z
 "$AWS_REGION" ]; then
    echo "ERROR: No AWS region configured."
    echo "Run 'aws configure set region <region>' or export AWS_DEFAULT_REGION."
    exit 1
fi

# ============================================================================
# Runtime selection
# ============================================================================

echo ""
echo "=========================================="
echo "AWS Lambda - Create Your First Function"
echo "=========================================="
echo ""
echo "Select a runtime for your Lambda function:"
echo "  1) Python 3.13"
echo "  2) Node.js 22.x"
echo ""
echo "Using default: Python 3.13"
RUNTIME_CHOICE="1"

case "$RUNTIME_CHOICE" in
```

```
    1)
        RUNTIME="python3.13"
        HANDLER="lambda_function.lambda_handler"
        CODE_FILE="lambda_function.py"
        cat > "${TEMP_DIR}/${CODE_FILE}" << 'PYTHON_EOF'
import json
import logging
logger = logging.getLogger()
logger.setLevel(logging.INFO)
def lambda_handler(event, context):
    if not isinstance(event, dict) or 'length' not in event or 'width' not in
 event:
        raise ValueError('Event must contain length and width')
    try:
        length = float(event['length'])
        width = float(event['width'])
        if length < 0 or width < 0:
            raise ValueError('Length and width must be non-negative')
        area = calculate_area(length, width)
        print(f'The area is {area}')
        logger.info(f'CloudWatch logs group: {context.log_group_name}')
        return json.dumps({'area': area})
    except (TypeError, ValueError) as e:
        logger.error(f'Error processing input: {str(e)}')
        raise
def calculate_area(length, width):
    return length * width
PYTHON_EOF
        echo "Selected runtime: Python 3.13"
        ;;
    2)
        RUNTIME="nodejs22.x"
        HANDLER="index.handler"
        CODE_FILE="index.mjs"
        cat > "${TEMP_DIR}/${CODE_FILE}" << 'NODEJS_EOF'
export const handler = async (event, context) => {
  if (!event || typeof event.length !== 'number' || typeof event.width !==
 'number') {
    throw new Error('Event must contain numeric length and width');
  }
  if (event.length < 0 || event.width < 0) {
    throw new Error('Length and width must be non-negative');
  }
  const area = event.length * event.width;
```

```
  console.log(`The area is ${area}`);
  console.log('CloudWatch log group: ', context.logGroupName);
  return JSON.stringify({area});
};
NODEJS_EOF
        echo "Selected runtime: Node.js 22.x"
        ;;
    *)
        echo "ERROR: Invalid choice. Please enter 1 or 2."
        exit 1
        ;;
esac

# ============================================================================
# Step 1: Create IAM execution role
# ============================================================================

echo ""
echo "=========================================="
echo "Step 1: Create IAM execution role"
echo "=========================================="
echo ""

TRUST_POLICY='{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "lambda.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}'

echo "Creating IAM role: ${ROLE_NAME}"
ROLE_OUTPUT=$(aws iam create-role \
    --role-name "$ROLE_NAME" \
    --assume-role-policy-document "$TRUST_POLICY" \
    --query 'Role.Arn' \
    --output text 2>&1)

if ! validate_input "$ROLE_OUTPUT" "^arn:aws:iam::[0-9]+:role/"; then
```

```
    echo "ERROR: Failed to create IAM role"
    exit 1
fi

echo "$ROLE_OUTPUT"
ROLE_ARN="$ROLE_OUTPUT"
CREATED_RESOURCES+=("iam-role:${ROLE_NAME}")
echo "Role ARN: ${ROLE_ARN}"

aws iam tag-role \
    --role-name "$ROLE_NAME" \
    --tags Key=project,Value=doc-smith Key=tutorial,Value=lambda-gettingstarted

echo ""
echo "Attaching AWSLambdaBasicExecutionRole policy..."
aws iam attach-role-policy \
    --role-name "$ROLE_NAME" \
    --policy-arn "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole" 2>&1
CREATED_RESOURCES+=("iam-role-policy:${ROLE_NAME}|arn:aws:iam::aws:policy/
service-role/AWSLambdaBasicExecutionRole")
echo "Policy attached."

# IAM roles can take a few seconds to propagate
echo "Waiting for IAM role to propagate..."
sleep 10

################################################################################
# Step 2: Create Lambda function
################################################################################

echo ""
echo "========================================="
echo "Step 2: Create Lambda function"
echo "========================================="
echo ""

echo "Creating deployment package..."
ORIGINAL_DIR=$(pwd)
cd "$TEMP_DIR" || exit 1
zip -j function.zip "$CODE_FILE" > /dev/null 2>&1 || {
    echo "ERROR: Failed to create deployment package"
    exit 1
}
```

```
cd "$ORIGINAL_DIR" || exit 1

if [ ! -f "${TEMP_DIR}/function.zip" ]; then
    echo "ERROR: Deployment package creation failed"
    exit 1
fi

echo "Creating Lambda function: ${FUNCTION_NAME}"
echo "  Runtime: ${RUNTIME}"
echo "  Handler: ${HANDLER}"
echo ""

CREATE_OUTPUT=$(aws lambda create-function \
    --function-name "$FUNCTION_NAME" \
    --runtime "$RUNTIME" \
    --role "$ROLE_ARN" \
    --handler "$HANDLER" \
    --architectures x86_64 \
    --zip-file "fileb://${TEMP_DIR}/function.zip" \
    --tags project=doc-smith,tutorial=lambda-gettingstarted \
    --query '[FunctionName, FunctionArn, Runtime, State]' \
    --output text 2>&1)

if [ -z "$CREATE_OUTPUT" ]; then
    echo "ERROR: Failed to create Lambda function"
    exit 1
fi

echo "$CREATE_OUTPUT"
CREATED_RESOURCES+=("lambda-function:${FUNCTION_NAME}")

wait_for_resource "Lambda function to become Active" \
    "aws lambda get-function-configuration --function-name ${FUNCTION_NAME} --
query State --output text" \
    "Active"

################################################################################
# Step 3: Invoke the function
################################################################################

echo ""
echo "============================================"
echo "Step 3: Invoke the function"
echo "============================================"
```

```
echo ""

TEST_EVENT='{"length": 6, "width": 7}'
echo "Invoking function with test event: ${TEST_EVENT}"
echo ""

echo "$TEST_EVENT" > "${TEMP_DIR}/test-event.json"

if ! validate_input "$TEST_EVENT" '"length": [0-9]+, "width": [0-9]+'; then
    echo "ERROR: Invalid test event format"
    exit 1
fi

INVOKE_OUTPUT=$(aws lambda invoke \
    --function-name "$FUNCTION_NAME" \
    --payload "fileb://${TEMP_DIR}/test-event.json" \
    --cli-read-timeout 30 \
    "${TEMP_DIR}/response.json" 2>&1)
echo "$INVOKE_OUTPUT"

if [ ! -f "${TEMP_DIR}/response.json" ]; then
    echo "ERROR: No response file generated"
    exit 1
fi

RESPONSE=$(cat "${TEMP_DIR}/response.json")
echo ""
echo "Function response: ${RESPONSE}"
echo ""

if echo "$INVOKE_OUTPUT" | grep -qi "functionerror"; then
    echo "WARNING: Function returned an error."
fi

################################################################################
# Step 4: View CloudWatch logs
################################################################################

echo ""
echo "========================================="
echo "Step 4: View CloudWatch Logs"
echo "========================================="
echo ""
```

```
echo "Log group: ${LOG_GROUP_NAME}"
echo ""

echo "Waiting for CloudWatch logs to be available..."

LOG_STREAMS=""
for i in $(seq 1 6); do
    LOG_STREAMS=$(aws logs describe-log-streams \
        --log-group-name "$LOG_GROUP_NAME" \
        --order-by LastEventTime \
        --descending \
        --query 'logStreams[0].logStreamName' \
        --output text 2>/dev/null) || true
    if [ -n "$LOG_STREAMS" ] && [ "$LOG_STREAMS" != "None" ]; then
        break
    fi
    LOG_STREAMS=""
    sleep 5
done

if [ -n "$LOG_STREAMS" ] && [ "$LOG_STREAMS" != "None" ]; then
    echo "Latest log stream: ${LOG_STREAMS}"
    echo ""
    echo "--- Log events ---"
    LOG_EVENTS=$(aws logs get-log-events \
        --log-group-name "$LOG_GROUP_NAME" \
        --log-stream-name "$LOG_STREAMS" \
        --query 'events[].message' \
        --output text 2>&1) || true
    echo "$LOG_EVENTS"
    echo "--- End of log events ---"
else
    echo "No log streams found yet. Logs may take a moment to appear."
    echo "You can view them in the CloudWatch console:"
    echo "  Log group: ${LOG_GROUP_NAME}"
fi

CREATED_RESOURCES+=("log-group:${LOG_GROUP_NAME}")

aws logs tag-log-group \
    --log-group-name "$LOG_GROUP_NAME" \
    --tags project=doc-smith,tutorial=lambda-gettingstarted

################################################################################
```

```
# Summary and cleanup
################################################################################

echo ""
echo "==========================================="
echo "SUMMARY"
echo "==========================================="
echo ""
echo "Resources created:"
echo "  IAM role:          ${ROLE_NAME}"
echo "  Lambda function:   ${FUNCTION_NAME}"
echo "  CloudWatch logs:   ${LOG_GROUP_NAME}"
echo ""
echo "==========================================="
echo "CLEANUP"
echo "==========================================="
echo ""
echo "Cleaning up all created resources..."
cleanup_resources

echo ""
echo "Done."
```

- Untuk detail API, lihat topik berikut di Referensi Perintah AWS CLI .
  - AttachRolePolicy
  - CreateFunction
  - CreateRole
  - DeleteFunction
  - DeleteLogGroup
  - DeleteRole
  - DescribeLogStreams
  - DetachRolePolicy
  - GetFunction
  - GetFunctionConfiguration
  - GetLogEvents
  - Memanggil

Untuk daftar lengkap AWS Panduan pengembang SDK dan contoh kode, lihat Menggunakan Lambda dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Menginvokasi fungsi Lambda dari browser

Contoh kode berikut menunjukkan cara memanggil AWS Lambda fungsi dari browser.

JavaScript

SDK untuk JavaScript (v2)

Anda dapat membuat aplikasi berbasis browser yang menggunakan AWS Lambda fungsi untuk memperbarui tabel Amazon DynamoDB dengan pilihan pengguna.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkap di GitHub.

Layanan yang digunakan dalam contoh ini

- DynamoDB
- Lambda

SDK untuk JavaScript (V3)

Anda dapat membuat aplikasi berbasis browser yang menggunakan AWS Lambda fungsi untuk memperbarui tabel Amazon DynamoDB dengan pilihan pengguna. Aplikasi ini menggunakan AWS SDK untuk JavaScript v3.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkap di GitHub.

Layanan yang digunakan dalam contoh ini

- DynamoDB
- Lambda

Untuk daftar lengkap AWS Panduan pengembang SDK dan contoh kode, lihat Menggunakan Lambda dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Ubah data untuk aplikasi Anda dengan S3 Object Lambda

Contoh kode berikut menunjukkan cara mengubah data untuk aplikasi Anda dengan S3 Object Lambda.

.NET

SDK untuk .NET

Menunjukkan cara menambahkan kode kustom ke permintaan GET S3 standar untuk memodifikasi objek yang diminta diambil dari S3 sehingga objek sesuai dengan kebutuhan klien atau aplikasi yang meminta.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkap di GitHub.

Layanan yang digunakan dalam contoh ini

- Lambda
- Amazon S3

Untuk daftar lengkap AWS Panduan pengembang SDK dan contoh kode, lihat Menggunakan Lambda dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Menggunakan API Gateway untuk menginvokasi fungsi Lambda

Contoh kode berikut menunjukkan cara membuat AWS Lambda fungsi yang dipanggil oleh Amazon API Gateway.

Java

SDK untuk Java 2.x

Menunjukkan cara membuat AWS Lambda berfungsi dengan menggunakan Lambda Java runtime API. Contoh ini memanggil yang berbeda AWS layanan untuk melakukan kasus penggunaan tertentu. Contoh ini menunjukkan cara membuat fungsi Lambda yang diinvokasi oleh Amazon API Gateway yang memindai peringatan hari jadi kerja di tabel Amazon DynamoDB dan menggunakan Amazon Simple Notification Service (Amazon SNS) untuk mengirim pesan teks berisi ucapan selamat kepada karyawan Anda pada tanggal hari jadi kerja satu tahun mereka.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkap di GitHub.

Layanan yang digunakan dalam contoh ini

- API Gateway
- DynamoDB
- Lambda
- Amazon SNS

JavaScript

SDK untuk JavaScript (V3)

Menunjukkan cara membuat AWS Lambda fungsi dengan menggunakan Lambda JavaScript API waktu proses. Contoh ini memanggil yang berbeda AWS layanan untuk melakukan kasus penggunaan tertentu. Contoh ini menunjukkan cara membuat fungsi Lambda yang diinvokasi oleh Amazon API Gateway yang memindai peringatan hari jadi kerja di tabel Amazon DynamoDB dan menggunakan Amazon Simple Notification Service (Amazon SNS) untuk mengirim pesan teks berisi ucapan selamat kepada karyawan Anda pada tanggal hari jadi kerja satu tahun mereka.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkap di GitHub.

Contoh ini juga tersedia di panduan developer AWS SDK untuk JavaScript v3.

Layanan yang digunakan dalam contoh ini

- API Gateway
- DynamoDB
- Lambda
- Amazon SNS

Python

SDK untuk Python (Boto3)

Contoh ini menunjukkan cara membuat dan menggunakan REST API API Gateway Amazon yang menargetkan AWS Lambda fungsi. Handler Lambda menunjukkan cara merutekan

berdasarkan metode HTTP; cara mendapatkan data dari string kueri, header, dan body; dan cara mengembalikan respons JSON.

- Terapkan fungsi Lambda.
- Buat API REST API Gateway.
- Buat sumber REST yang menargetkan fungsi Lambda.
- Berikan izin untuk membiarkan API Gateway memanggil fungsi Lambda.
- Gunakan paket Permintaan untuk mengirim permintaan ke REST API.
- Bersihkan semua sumber daya yang dibuat selama demo.

Contoh ini paling baik dilihat pada GitHub. Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkap di GitHub.

Layanan yang digunakan dalam contoh ini

- API Gateway
- DynamoDB
- Lambda
- Amazon SNS

Untuk daftar lengkap AWS Panduan pengembang SDK dan contoh kode, lihat Menggunakan Lambda dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Menggunakan Step Functions untuk menginvokasi fungsi Lambda

Contoh kode berikut menunjukkan cara membuat AWS Step Functions mesin negara yang memanggil AWS Lambda fungsi secara berurutan.

Java

SDK untuk Java 2.x

Menunjukkan cara membuat AWS alur kerja tanpa server dengan menggunakan AWS Step Functions dan AWS SDK for Java 2.x. Setiap langkah alur kerja diimplementasikan menggunakan AWS Lambda fungsi.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkap di GitHub.

Layanan yang digunakan dalam contoh ini

- DynamoDB
- Lambda
- Amazon SES
- Fungsi Langkah

Untuk daftar lengkap AWS Panduan pengembang SDK dan contoh kode, lihat Menggunakan Lambda dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Menggunakan peristiwa terjadwal untuk menginvokasi fungsi Lambda

Contoh kode berikut menunjukkan cara membuat AWS Lambda fungsi yang dipanggil oleh Amazon EventBridge acara yang dijadwalkan.

Java

SDK untuk Java 2.x

Menunjukkan cara membuat Amazon EventBridge acara terjadwal yang memanggil AWS Lambda fungsi. Konfigurasikan EventBridge untuk menggunakan ekspresi cron untuk menjadwalkan ketika fungsi Lambda dipanggil. Dalam contoh ini, Anda membuat fungsi Lambda menggunakan API runtime Java Lambda. Contoh ini memanggil yang berbeda AWS layanan untuk melakukan kasus penggunaan tertentu. Contoh ini menunjukkan cara membuat aplikasi yang mengirimkan pesan teks seluler kepada karyawan Anda berisi ucapan selamat pada hari jadi setahun kerja mereka.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkap di GitHub.

Layanan yang digunakan dalam contoh ini

- CloudWatch Log
- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

JavaScript

SDK untuk JavaScript (V3)

Menunjukkan cara membuat Amazon EventBridge acara terjadwal yang memanggil AWS Lambda fungsi. Konfigurasikan EventBridge untuk menggunakan ekspresi cron untuk menjadwalkan ketika fungsi Lambda dipanggil. Dalam contoh ini, Anda membuat fungsi Lambda dengan menggunakan Lambda JavaScript API waktu proses. Contoh ini memanggil yang berbeda AWS layanan untuk melakukan kasus penggunaan tertentu. Contoh ini menunjukkan cara membuat aplikasi yang mengirimkan pesan teks seluler kepada karyawan Anda berisi ucapan selamat pada hari jadi setahun kerja mereka.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkap di GitHub.

Contoh ini juga tersedia di panduan developer AWS SDK untuk JavaScript v3.

Layanan yang digunakan dalam contoh ini

- CloudWatch Log
- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

Python

SDK untuk Python (Boto3)

Contoh ini menunjukkan cara mendaftarkan AWS Lambda berfungsi sebagai target Amazon yang dijadwalkan EventBridge acara. Penangan Lambda menulis pesan ramah dan data acara lengkap ke Amazon CloudWatch Log untuk pengambilan nanti.

- Menyebarkan fungsi Lambda.
- Menciptakan sebuah EventBridge acara terjadwal dan menjadikan fungsi Lambda sebagai target.
- Memberikan izin untuk menyewa EventBridge memanggil fungsi Lambda.
- Mencetak data terbaru dari CloudWatch Log untuk menunjukkan hasil pemanggilan yang dijadwalkan.

- Membersihkan semua sumber daya yang dibuat selama demo.

Contoh ini paling baik dilihat pada GitHub. Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkap di GitHub.

Layanan yang digunakan dalam contoh ini

- CloudWatch Log
- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

Untuk daftar lengkap AWS Panduan pengembang SDK dan contoh kode, lihat Menggunakan Lambda dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Gunakan Amazon Neptune API untuk mengembangkan fungsi Lambda yang menanyakan data grafik

Contoh kode berikut menunjukkan cara menggunakan Neptune API untuk menanyakan data grafik.

Java

SDK untuk Java 2.x

Menunjukkan cara menggunakan Amazon Neptune Java API untuk membuat fungsi Lambda yang menanyakan data grafik dalam VPC.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkap di GitHub.

Layanan yang digunakan dalam contoh ini

- Lambda
- Neptune

Untuk daftar lengkap AWS Panduan pengembang SDK dan contoh kode, lihat Menggunakan Lambda dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Menggunakan variabel properti dalam dasbor pemantauan untuk memantau beberapa fungsi tanpa server

Contoh kode berikut ini menunjukkan cara untuk melakukan:

- Buat fungsi Lambda untuk pemantauan
- Buat CloudWatch dasbor
- Menambahkan variabel properti ke dasbor
- Pembersihan sumber daya

Bash

AWS CLI dengan skrip Bash

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankan di Contoh tutorial pengembang repositori.

```
#!/bin/bash

# CloudWatch Dashboard with Lambda Function Variable Script
# This script creates a CloudWatch dashboard with a property variable for Lambda
 function names

set -euo pipefail

# Security: Set restrictive umask
umask 0077

# Set up logging with secure permissions
LOG_FILE="cloudwatch-dashboard-script-v4.log"
touch "$LOG_FILE"
chmod 600 "$LOG_FILE"
echo "Starting script execution at $(date)" >> "$LOG_FILE"

# Function to log commands and their output (with sensitive data sanitization)
log_cmd() {
```

```
    local cmd="$1"
    local sanitized_cmd="${cmd//--password*/--password [REDACTED]}"
    sanitized_cmd="${sanitized_cmd//--secret*/--secret [REDACTED]}"
    echo "$(date): Running command: $sanitized_cmd" >> "$LOG_FILE"
    eval "$cmd" 2>&1 | tee -a "$LOG_FILE"
    return ${PIPESTATUS[0]}
}

# Function to check for errors in command output
check_error() {
    local cmd_output="$1"
    local cmd_status="$2"
    local error_msg="$3"

    if [ $cmd_status -ne 0 ] || echo "$cmd_output" | grep -qi "error"; then
        echo "ERROR: $error_msg" | tee -a "$LOG_FILE"
        # Sanitize output before logging
        local sanitized_output="${cmd_output//arn:aws:iam::[0-9]*/
arn:aws:iam::ACCOUNT_ID}"
        echo "Command output: $sanitized_output" | tee -a "$LOG_FILE"
        cleanup_resources
        exit 1
    fi
}

# Trap errors and cleanup
trap 'cleanup_resources' EXIT ERR INT TERM

# Function to clean up resources
cleanup_resources() {
    local exit_code=$?

    echo "" | tee -a "$LOG_FILE"
    echo "========================================" | tee -a "$LOG_FILE"
    echo "CLEANUP PROCESS" | tee -a "$LOG_FILE"
    echo "========================================" | tee -a "$LOG_FILE"

    if [ -n "${DASHBOARD_NAME:-}" ]; then
        echo "Deleting CloudWatch dashboard: $DASHBOARD_NAME" | tee -a
 "$LOG_FILE"
        aws cloudwatch delete-dashboards --dashboard-names "$DASHBOARD_NAME" 2>&1
 >> "$LOG_FILE" || true
    fi
```

```
    if [ -n "${LAMBDA_FUNCTION1:-}" ]; then
        echo "Deleting Lambda function: $LAMBDA_FUNCTION1" | tee -a "$LOG_FILE"
        aws lambda delete-function --function-name "$LAMBDA_FUNCTION1" 2>&1 >>
 "$LOG_FILE" || true
    fi

    if [ -n "${LAMBDA_FUNCTION2:-}" ]; then
        echo "Deleting Lambda function: $LAMBDA_FUNCTION2" | tee -a "$LOG_FILE"
        aws lambda delete-function --function-name "$LAMBDA_FUNCTION2" 2>&1 >>
 "$LOG_FILE" || true
    fi

    if [ -n "${ROLE_NAME:-}" ]; then
        echo "Detaching policy from role: $ROLE_NAME" | tee -a "$LOG_FILE"
        aws iam detach-role-policy --role-name "$ROLE_NAME" --policy-arn
 arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole 2>&1 >>
 "$LOG_FILE" || true

        echo "Deleting IAM role: $ROLE_NAME" | tee -a "$LOG_FILE"
        aws iam delete-role --role-name "$ROLE_NAME" 2>&1 >> "$LOG_FILE" || true
    fi

    # Clean up temporary files securely
    shred -vfz -n 3 trust-policy.json lambda_function.py lambda_function.zip 2>/
dev/null || rm -f trust-policy.json lambda_function.py lambda_function.zip

    echo "Cleanup completed." | tee -a "$LOG_FILE"

    return $exit_code
}

# Validate AWS CLI is installed and authenticated
if ! command -v aws &> /dev/null; then
    echo "ERROR: AWS CLI is not installed" | tee -a "$LOG_FILE"
    exit 1
fi

if ! aws sts get-caller-identity &> /dev/null; then
    echo "ERROR: AWS CLI is not properly authenticated" | tee -a "$LOG_FILE"
    exit 1
fi

# Get AWS region with validation
AWS_REGION=$(aws configure get region 2>/dev/null || echo "")
```

```
if [ -z "$AWS_REGION" ]; then
    AWS_REGION="us-east-1"
    echo "No region found in AWS config, defaulting to $AWS_REGION" | tee -a
 "$LOG_FILE"
else
    echo "Using AWS region: $AWS_REGION" | tee -a "$LOG_FILE"
fi

# Validate region format
if ! [[ "$AWS_REGION" =~ ^[a-z]{2}-[a-z]+-[0-9]$ ]]; then
    echo "ERROR: Invalid AWS region format: $AWS_REGION" | tee -a "$LOG_FILE"
    exit 1
fi

# Generate unique identifiers using secure random with validation
RANDOM_ID=$(openssl rand -hex 6)
if [ -z "$RANDOM_ID" ] || [ ${#RANDOM_ID} -ne 12 ]; then
    echo "ERROR: Failed to generate valid random identifier" | tee -a "$LOG_FILE"
    exit 1
fi

DASHBOARD_NAME="LambdaMetricsDashboard-${RANDOM_ID}"
LAMBDA_FUNCTION1="TestFunction1-${RANDOM_ID}"
LAMBDA_FUNCTION2="TestFunction2-${RANDOM_ID}"
ROLE_NAME="LambdaExecutionRole-${RANDOM_ID}"

# Validate resource names don't exceed AWS limits
if [ ${#DASHBOARD_NAME} -gt 128 ] || [ ${#LAMBDA_FUNCTION1} -gt 64 ] ||
 [ ${#ROLE_NAME} -gt 64 ]; then
    echo "ERROR: Generated resource names exceed AWS limits" | tee -a "$LOG_FILE"
    exit 1
fi

echo "Using random identifier: $RANDOM_ID" | tee -a "$LOG_FILE"
echo "Dashboard name: $DASHBOARD_NAME" | tee -a "$LOG_FILE"
echo "Lambda function names: $LAMBDA_FUNCTION1, $LAMBDA_FUNCTION2" | tee -a
 "$LOG_FILE"
echo "IAM role name: $ROLE_NAME" | tee -a "$LOG_FILE"

# Create IAM role for Lambda functions
echo "Creating IAM role for Lambda..." | tee -a "$LOG_FILE"
TRUST_POLICY='{
  "Version":"2012-10-17",
  "Statement": [
```

```
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "lambda.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}'

echo "$TRUST_POLICY" > trust-policy.json
chmod 600 trust-policy.json

# Validate JSON before use
if ! python3 -m json.tool trust-policy.json > /dev/null 2>&1; then
    echo "ERROR: Invalid trust policy JSON" | tee -a "$LOG_FILE"
    exit 1
fi

ROLE_OUTPUT=$(log_cmd "aws iam create-role --role-name '$ROLE_NAME' --assume-
role-policy-document file://trust-policy.json --tags Key=project,Value=doc-smith
 Key=tutorial,Value=cloudwatch-streams --output json")
check_error "$ROLE_OUTPUT" $? "Failed to create IAM role"

ROLE_ARN=$(echo "$ROLE_OUTPUT" | python3 -c "import sys, json;
 print(json.load(sys.stdin)['Role']['Arn'])" 2>/dev/null)
if [ -z "$ROLE_ARN" ]; then
    echo "ERROR: Failed to extract Role ARN" | tee -a "$LOG_FILE"
    exit 1
fi
echo "Role ARN: $ROLE_ARN" | tee -a "$LOG_FILE"

# Attach Lambda basic execution policy to the role
echo "Attaching Lambda execution policy to role..." | tee -a "$LOG_FILE"
POLICY_OUTPUT=$(log_cmd "aws iam attach-role-policy --role-name '$ROLE_NAME' --
policy-arn arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole")
check_error "$POLICY_OUTPUT" $? "Failed to attach policy to role"

# Wait for role to propagate
echo "Waiting for IAM role to propagate..." | tee -a "$LOG_FILE"
sleep 10

# Create simple Python Lambda function code with security validation
echo "Creating Lambda function code..." | tee -a "$LOG_FILE"
```

```
cat > lambda_function.py << 'LAMBDA_EOF'
def handler(event, context):
    print("Lambda function executed successfully")
    return {
        'statusCode': 200,
        'body': 'Success'
    }
LAMBDA_EOF

chmod 600 lambda_function.py

# Validate Python syntax
if ! python3 -m py_compile lambda_function.py 2>/dev/null; then
    echo "ERROR: Invalid Python syntax in Lambda function" | tee -a "$LOG_FILE"
    exit 1
fi

# Zip the Lambda function code
zip -j -q lambda_function.zip lambda_function.py
if [ ! -f lambda_function.zip ]; then
    echo "ERROR: Failed to create lambda_function.zip" | tee -a "$LOG_FILE"
    exit 1
fi
chmod 600 lambda_function.zip

# Validate zip file integrity
if ! unzip -t lambda_function.zip > /dev/null 2>&1; then
    echo "ERROR: Created zip file is corrupted" | tee -a "$LOG_FILE"
    exit 1
fi

# Create first Lambda function
echo "Creating first Lambda function: $LAMBDA_FUNCTION1..." | tee -a "$LOG_FILE"
LAMBDA1_OUTPUT=$(log_cmd "aws lambda create-function --function-name
 '$LAMBDA_FUNCTION1' --runtime python3.11 --role '$ROLE_ARN' --handler
 lambda_function.handler --zip-file fileb://lambda_function.zip --timeout 30 --
memory-size 128 --tags project=doc-smith,tutorial=cloudwatch-streams")
check_error "$LAMBDA1_OUTPUT" $? "Failed to create first Lambda function"

# Create second Lambda function
echo "Creating second Lambda function: $LAMBDA_FUNCTION2..." | tee -a "$LOG_FILE"
LAMBDA2_OUTPUT=$(log_cmd "aws lambda create-function --function-name
 '$LAMBDA_FUNCTION2' --runtime python3.11 --role '$ROLE_ARN' --handler
```

```
 lambda_function.handler --zip-file fileb://lambda_function.zip --timeout 30 --
memory-size 128 --tags project=doc-smith,tutorial=cloudwatch-streams")
check_error "$LAMBDA2_OUTPUT" $? "Failed to create second Lambda function"

# Invoke Lambda functions to generate some metrics
echo "Invoking Lambda functions to generate metrics..." | tee -a "$LOG_FILE"
log_cmd "aws lambda invoke --function-name '$LAMBDA_FUNCTION1' --payload '{}' /
dev/null" || true
log_cmd "aws lambda invoke --function-name '$LAMBDA_FUNCTION2' --payload '{}' /
dev/null" || true

# Create CloudWatch dashboard with property variable
echo "Creating CloudWatch dashboard with property variable..." | tee -a
 "$LOG_FILE"

# Create dashboard body with proper escaping and validation
DASHBOARD_BODY=$(cat <<EOF
{
  "widgets": [
    {
      "type": "metric",
      "x": 0,
      "y": 0,
      "width": 12,
      "height": 6,
      "properties": {
        "metrics": [
          [ "AWS/Lambda", "Invocations", "FunctionName", "$LAMBDA_FUNCTION1" ]
        ],
        "view": "timeSeries",
        "stacked": false,
        "region": "$AWS_REGION",
        "title": "Lambda Invocations",
        "period": 300,
        "stat": "Sum"
      }
    }
  ]
}
EOF
)

# Validate JSON before sending
if ! echo "$DASHBOARD_BODY" | python3 -m json.tool > /dev/null 2>&1; then
```

```
    echo "ERROR: Dashboard body is not valid JSON" | tee -a "$LOG_FILE"
    exit 1
fi

# First create a basic dashboard without variables
echo "Creating initial dashboard without variables..." | tee -a "$LOG_FILE"
DASHBOARD_OUTPUT=$(aws cloudwatch put-dashboard --dashboard-name
 "$DASHBOARD_NAME" --dashboard-body "$DASHBOARD_BODY" --output json 2>&1)
check_error "$DASHBOARD_OUTPUT" $? "Failed to create initial CloudWatch
 dashboard"

# Now let's try to add a property variable using the console instructions
echo "To complete the tutorial, please follow these steps in the CloudWatch
 console:" | tee -a "$LOG_FILE"
echo "1. Open the CloudWatch console at https://console.aws.amazon.com/
cloudwatch/" | tee -a "$LOG_FILE"
echo "2. Navigate to Dashboards and select your dashboard: $DASHBOARD_NAME" | tee
 -a "$LOG_FILE"
echo "3. Choose Actions > Variables > Create a variable" | tee -a "$LOG_FILE"
echo "4. Choose Property variable" | tee -a "$LOG_FILE"
echo "5. For Property that the variable changes, choose FunctionName" | tee -a
 "$LOG_FILE"
echo "6. For Input type, choose Select menu (dropdown)" | tee -a "$LOG_FILE"
echo "7. Choose Use the results of a metric search" | tee -a "$LOG_FILE"
echo "8. Choose Pre-built queries > Lambda > Errors" | tee -a "$LOG_FILE"
echo "9. Choose By Function Name and then choose Search" | tee -a "$LOG_FILE"
echo "10. (Optional) Configure any secondary settings as desired" | tee -a
 "$LOG_FILE"
echo "11. Choose Add variable" | tee -a "$LOG_FILE"
echo "" | tee -a "$LOG_FILE"
echo "The dashboard has been created and can be accessed at:" | tee -a
 "$LOG_FILE"
echo "https://console.aws.amazon.com/cloudwatch/home#dashboards:name=
$DASHBOARD_NAME" | tee -a "$LOG_FILE"

# Verify dashboard creation
echo "Verifying dashboard creation..." | tee -a "$LOG_FILE"
VERIFY_OUTPUT=$(aws cloudwatch get-dashboard --dashboard-name "$DASHBOARD_NAME"
 --output json 2>&1)
check_error "$VERIFY_OUTPUT" $? "Failed to verify dashboard creation"

echo "" | tee -a "$LOG_FILE"
echo "========================================" | tee -a "$LOG_FILE"
echo "DASHBOARD CREATED SUCCESSFULLY" | tee -a "$LOG_FILE"
```

```
echo "========================================" | tee -a "$LOG_FILE"
echo "Dashboard Name: $DASHBOARD_NAME" | tee -a "$LOG_FILE"
echo "Lambda Functions: $LAMBDA_FUNCTION1, $LAMBDA_FUNCTION2" | tee -a
 "$LOG_FILE"
echo "" | tee -a "$LOG_FILE"
echo "You can view your dashboard in the CloudWatch console:" | tee -a
 "$LOG_FILE"
echo "https://console.aws.amazon.com/cloudwatch/home#dashboards:name=
$DASHBOARD_NAME" | tee -a "$LOG_FILE"
echo "" | tee -a "$LOG_FILE"

# Auto-confirm cleanup
echo "" | tee -a "$LOG_FILE"
echo "========================================" | tee -a "$LOG_FILE"
echo "CLEANUP CONFIRMATION" | tee -a "$LOG_FILE"
echo "========================================" | tee -a "$LOG_FILE"
echo "The following resources were created:" | tee -a "$LOG_FILE"
echo "- CloudWatch Dashboard: $DASHBOARD_NAME" | tee -a "$LOG_FILE"
echo "- Lambda Function: $LAMBDA_FUNCTION1" | tee -a "$LOG_FILE"
echo "- Lambda Function: $LAMBDA_FUNCTION2" | tee -a "$LOG_FILE"
echo "- IAM Role: $ROLE_NAME" | tee -a "$LOG_FILE"
echo "" | tee -a "$LOG_FILE"
echo "Auto-confirming cleanup of all created resources..." | tee -a "$LOG_FILE"

echo "Script completed successfully." | tee -a "$LOG_FILE"
exit 0
```

- Untuk detail API, lihat topik berikut di Referensi Perintah AWS CLI .
  - AttachRolePolicy
  - CreateFunction
  - CreateRole
  - DeleteDashboards
  - DeleteFunction
  - DeleteRole
  - DetachRolePolicy
  - GetDashboard
  - Memanggil
  - PutDashboard

Untuk daftar lengkap AWS Panduan pengembang SDK dan contoh kode, lihat Menggunakan Lambda dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Tulis data aktivitas khusus dengan fungsi Lambda setelah otentikasi pengguna Amazon Cognito menggunakan AWS SDK

Contoh kode berikut menunjukkan cara menulis data aktivitas khusus dengan fungsi Lambda setelah otentikasi pengguna Amazon Cognito.

- Gunakan fungsi administrator untuk menambahkan pengguna ke kumpulan pengguna.
- Konfigurasikan kumpulan pengguna untuk memanggil fungsi Lambda untuk `PostAuthentication` pemicu.
- Masuk pengguna baru ke Amazon Cognito.
- Fungsi Lambda menulis informasi khusus ke CloudWatch Log dan ke tabel DynamoDB.
- Dapatkan dan tampilkan data khusus dari tabel DynamoDB, lalu bersihkan sumber daya.

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkap dan pelajari cara menyiapkan dan menjalankan di Repositori Contoh Kode AWS.

Jalankan skenario interaktif di penggugah/prompt perintah.

```
import (
 "context"
 "errors"
 "log"
 "strings"
 "user_pools_and_lambda_triggers/actions"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
```

```
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

// ActivityLog separates the steps of this scenario into individual functions so
 that
// they are simpler to read and understand.
type ActivityLog struct {
 helper       IScenarioHelper
 questioner   demotools.IQuestioner
 resources    Resources
 cognitoActor *actions.CognitoActions
}

// NewActivityLog constructs a new activity log runner.
func NewActivityLog(sdkConfig aws.Config, questioner demotools.IQuestioner,
 helper IScenarioHelper) ActivityLog {
 scenario := ActivityLog{
  helper:       helper,
  questioner:   questioner,
  resources:    Resources{},
  cognitoActor: &actions.CognitoActions{CognitoClient:
 cognitoidentityprovider.NewFromConfig(sdkConfig)},
 }
 scenario.resources.init(scenario.cognitoActor, questioner)
 return scenario
}

// AddUserToPool selects a user from the known users table and uses administrator
 credentials to add the user to the user pool.
func (runner *ActivityLog) AddUserToPool(ctx context.Context, userPoolId string,
 tableName string) (string, string) {
 log.Println("To facilitate this example, let's add a user to the user pool using
 administrator privileges.")
 users, err := runner.helper.GetKnownUsers(ctx, tableName)
 if err != nil {
  panic(err)
 }
 user := users.Users[0]
 log.Printf("Adding known user %v to the user pool.\n", user.UserName)
 err = runner.cognitoActor.AdminCreateUser(ctx, userPoolId, user.UserName,
 user.UserEmail)
 if err != nil {
  panic(err)
```

```
 }
 pwSet := false
 password := runner.questioner.AskPassword("\nEnter a password that has at least
 eight characters, uppercase, lowercase, numbers and symbols.\n"+
  "(the password will not display as you type):", 8)
 for !pwSet {
  log.Printf("\nSetting password for user '%v'.\n", user.UserName)
  err = runner.cognitoActor.AdminSetUserPassword(ctx, userPoolId, user.UserName,
 password)
  if err != nil {
   var invalidPassword *types.InvalidPasswordException
   if errors.As(err, &invalidPassword) {
    password = runner.questioner.AskPassword("\nEnter another password:", 8)
   } else {
    panic(err)
   }
  } else {
   pwSet = true
  }
 }

 log.Println(strings.Repeat("-", 88))

 return user.UserName, password
}

// AddActivityLogTrigger adds a Lambda handler as an invocation target for the
 PostAuthentication trigger.
func (runner *ActivityLog) AddActivityLogTrigger(ctx context.Context, userPoolId
 string, activityLogArn string) {
 log.Println("Let's add a Lambda function to handle the PostAuthentication
 trigger from Cognito.\n" +
  "This trigger happens after a user is authenticated, and lets your function
 take action, such as logging\n" +
  "the outcome.")
 err := runner.cognitoActor.UpdateTriggers(
  ctx, userPoolId,
  actions.TriggerInfo{Trigger: actions.PostAuthentication, HandlerArn:
 aws.String(activityLogArn)})
 if err != nil {
  panic(err)
 }
 runner.resources.triggers = append(runner.resources.triggers,
 actions.PostAuthentication)
```

```
 log.Printf("Lambda function %v added to user pool %v to handle
 PostAuthentication Cognito trigger.\n",
  activityLogArn, userPoolId)

 log.Println(strings.Repeat("-", 88))
}

// SignInUser signs in as the specified user.
func (runner *ActivityLog) SignInUser(ctx context.Context, clientId string,
 userName string, password string) {
 log.Printf("Now we'll sign in user %v and check the results in the logs and the
 DynamoDB table.", userName)
 runner.questioner.Ask("Press Enter when you're ready.")
 authResult, err := runner.cognitoActor.SignIn(ctx, clientId, userName, password)
 if err != nil {
  panic(err)
 }
 log.Println("Sign in successful.",
  "The PostAuthentication Lambda handler writes custom information to CloudWatch
 Logs.")

 runner.resources.userAccessTokens = append(runner.resources.userAccessTokens,
 *authResult.AccessToken)
}

// GetKnownUserLastLogin gets the login info for a user from the Amazon DynamoDB
 table and displays it.
func (runner *ActivityLog) GetKnownUserLastLogin(ctx context.Context, tableName
 string, userName string) {
 log.Println("The PostAuthentication handler also writes login data to the
 DynamoDB table.")
 runner.questioner.Ask("Press Enter when you're ready to continue.")
 users, err := runner.helper.GetKnownUsers(ctx, tableName)
 if err != nil {
  panic(err)
 }
 for _, user := range users.Users {
  if user.UserName == userName {
   log.Println("The last login info for the user in the known users table is:")
   log.Printf("\t%+v", *user.LastLogin)
  }
 }
 log.Println(strings.Repeat("-", 88))
}
```

```
// Run runs the scenario.
func (runner *ActivityLog) Run(ctx context.Context, stackName string) {
 defer func() {
  if r := recover(); r != nil {
   log.Println("Something went wrong with the demo.")
   runner.resources.Cleanup(ctx)
  }
 }()

 log.Println(strings.Repeat("-", 88))
 log.Printf("Welcome\n")

 log.Println(strings.Repeat("-", 88))

 stackOutputs, err := runner.helper.GetStackOutputs(ctx, stackName)
 if err != nil {
  panic(err)
 }
 runner.resources.userPoolId = stackOutputs["UserPoolId"]
 runner.helper.PopulateUserTable(ctx, stackOutputs["TableName"])
 userName, password := runner.AddUserToPool(ctx, stackOutputs["UserPoolId"],
 stackOutputs["TableName"])

 runner.AddActivityLogTrigger(ctx, stackOutputs["UserPoolId"],
 stackOutputs["ActivityLogFunctionArn"])
 runner.SignInUser(ctx, stackOutputs["UserPoolClientId"], userName, password)
 runner.helper.ListRecentLogEvents(ctx, stackOutputs["ActivityLogFunction"])
 runner.GetKnownUserLastLogin(ctx, stackOutputs["TableName"], userName)

 runner.resources.Cleanup(ctx)

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}
```

Tangani `PostAuthentication` pemicu dengan fungsi Lambda.

```
import (
```

```
 "context"
 "fmt"
 "log"
 "os"
 "time"

 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 dynamodbtypes "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

const TABLE_NAME = "TABLE_NAME"

// LoginInfo defines structured login data that can be marshalled to a DynamoDB
 format.
type LoginInfo struct {
 UserPoolId string `dynamodbav:"UserPoolId"`
 ClientId   string `dynamodbav:"ClientId"`
 Time       string `dynamodbav:"Time"`
}

// UserInfo defines structured user data that can be marshalled to a DynamoDB
 format.
type UserInfo struct {
 UserName  string    `dynamodbav:"UserName"`
 UserEmail string    `dynamodbav:"UserEmail"`
 LastLogin LoginInfo `dynamodbav:"LastLogin"`
}

// GetKey marshals the user email value to a DynamoDB key format.
func (user UserInfo) GetKey() map[string]dynamodbtypes.AttributeValue {
 userEmail, err := attributevalue.Marshal(user.UserEmail)
 if err != nil {
  panic(err)
 }
 return map[string]dynamodbtypes.AttributeValue{"UserEmail": userEmail}
}

type handler struct {
 dynamoClient *dynamodb.Client
```

```
}

// HandleRequest handles the PostAuthentication event by writing custom data to
 the logs and
// to an Amazon DynamoDB table.
func (h *handler) HandleRequest(ctx context.Context,
 event events.CognitoEventUserPoolsPostAuthentication)
 (events.CognitoEventUserPoolsPostAuthentication, error) {
 log.Printf("Received post authentication trigger from %v for user '%v'",
 event.TriggerSource, event.UserName)
 tableName := os.Getenv(TABLE_NAME)
 user := UserInfo{
  UserName:  event.UserName,
  UserEmail: event.Request.UserAttributes["email"],
  LastLogin: LoginInfo{
   UserPoolId: event.UserPoolID,
   ClientId:   event.CallerContext.ClientID,
   Time:       time.Now().Format(time.UnixDate),
  },
 }
 // Write to CloudWatch Logs.
 fmt.Printf("%#v", user)

 // Also write to an external system. This examples uses DynamoDB to demonstrate.
 userMap, err := attributevalue.MarshalMap(user)
 if err != nil {
  log.Printf("Couldn't marshal to DynamoDB map. Here's why: %v\n", err)
 } else if len(userMap) == 0 {
  log.Printf("User info marshaled to an empty map.")
 } else {
  _, err := h.dynamoClient.PutItem(ctx, &dynamodb.PutItemInput{
   Item:      userMap,
   TableName: aws.String(tableName),
  })
  if err != nil {
   log.Printf("Couldn't write to DynamoDB. Here's why: %v\n", err)
  } else {
   log.Printf("Wrote user info to DynamoDB table %v.\n", tableName)
  }
 }

 return event, nil
}
```

```
func main() {
 ctx := context.Background()
 sdkConfig, err := config.LoadDefaultConfig(ctx)
 if err != nil {
  log.Panicln(err)
 }
 h := handler{
  dynamoClient: dynamodb.NewFromConfig(sdkConfig),
 }
 lambda.Start(h.HandleRequest)
}
```

Buat struct yang melakukan tugas-tugas umum.

```
import (
 "context"
 "log"
 "strings"
 "time"
 "user_pools_and_lambda_triggers/actions"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cloudformation"
 "github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

// IScenarioHelper defines common functions used by the workflows in this
 example.
type IScenarioHelper interface {
 Pause(secs int)
 GetStackOutputs(ctx context.Context, stackName string) (actions.StackOutputs,
 error)
 PopulateUserTable(ctx context.Context, tableName string)
 GetKnownUsers(ctx context.Context, tableName string) (actions.UserList, error)
 AddKnownUser(ctx context.Context, tableName string, user actions.User)
 ListRecentLogEvents(ctx context.Context, functionName string)
}
```

```
// ScenarioHelper contains AWS wrapper structs used by the workflows in this
 example.
type ScenarioHelper struct {
 questioner  demotools.IQuestioner
 dynamoActor *actions.DynamoActions
 cfnActor    *actions.CloudFormationActions
 cwlActor    *actions.CloudWatchLogsActions
 isTestRun   bool
}

// NewScenarioHelper constructs a new scenario helper.
func NewScenarioHelper(sdkConfig aws.Config, questioner demotools.IQuestioner)
 ScenarioHelper {
 scenario := ScenarioHelper{
  questioner:  questioner,
  dynamoActor: &actions.DynamoActions{DynamoClient:
 dynamodb.NewFromConfig(sdkConfig)},
  cfnActor:    &actions.CloudFormationActions{CfnClient:
 cloudformation.NewFromConfig(sdkConfig)},
  cwlActor:    &actions.CloudWatchLogsActions{CwlClient:
 cloudwatchlogs.NewFromConfig(sdkConfig)},
 }
 return scenario
}

// Pause waits for the specified number of seconds.
func (helper ScenarioHelper) Pause(secs int) {
 if !helper.isTestRun {
  time.Sleep(time.Duration(secs) * time.Second)
 }
}

// GetStackOutputs gets the outputs from the specified CloudFormation stack in a
 structured format.
func (helper ScenarioHelper) GetStackOutputs(ctx context.Context, stackName
 string) (actions.StackOutputs, error) {
 return helper.cfnActor.GetOutputs(ctx, stackName), nil
}

// PopulateUserTable fills the known user table with example data.
func (helper ScenarioHelper) PopulateUserTable(ctx context.Context, tableName
 string) {
 log.Printf("First, let's add some users to the DynamoDB %v table we'll use for
 this example.\n", tableName)
```

```
 err := helper.dynamoActor.PopulateTable(ctx, tableName)
 if err != nil {
  panic(err)
 }
}

// GetKnownUsers gets the users from the known users table in a structured
 format.
func (helper ScenarioHelper) GetKnownUsers(ctx context.Context, tableName string)
 (actions.UserList, error) {
 knownUsers, err := helper.dynamoActor.Scan(ctx, tableName)
 if err != nil {
  log.Printf("Couldn't get known users from table %v. Here's why: %v\n",
 tableName, err)
 }
 return knownUsers, err
}

// AddKnownUser adds a user to the known users table.
func (helper ScenarioHelper) AddKnownUser(ctx context.Context, tableName string,
 user actions.User) {
 log.Printf("Adding user '%v' with email '%v' to the DynamoDB known users
 table...\n",
  user.UserName, user.UserEmail)
 err := helper.dynamoActor.AddUser(ctx, tableName, user)
 if err != nil {
  panic(err)
 }
}

// ListRecentLogEvents gets the most recent log stream and events for the
 specified Lambda function and displays them.
func (helper ScenarioHelper) ListRecentLogEvents(ctx context.Context,
 functionName string) {
 log.Println("Waiting a few seconds to let Lambda write to CloudWatch Logs...")
 helper.Pause(10)
 log.Println("Okay, let's check the logs to find what's happened recently with
 your Lambda function.")
 logStream, err := helper.cwlActor.GetLatestLogStream(ctx, functionName)
 if err != nil {
  panic(err)
 }
 log.Printf("Getting some recent events from log stream %v\n",
 *logStream.LogStreamName)
```

```
 events, err := helper.cwlActor.GetLogEvents(ctx, functionName,
 *logStream.LogStreamName, 10)
 if err != nil {
  panic(err)
 }
 for _, event := range events {
  log.Printf("\t%v", *event.Message)
 }
 log.Println(strings.Repeat("-", 88))
}
```

Buat struct yang membungkus tindakan Amazon Cognito.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
)

type CognitoActions struct {
 CognitoClient *cognitoidentityprovider.Client
}



// Trigger and TriggerInfo define typed data for updating an Amazon Cognito
 trigger.
type Trigger int

const (
 PreSignUp Trigger = iota
 UserMigration
 PostAuthentication
)

type TriggerInfo struct {
```

```
 Trigger    Trigger
 HandlerArn *string
}

// UpdateTriggers adds or removes Lambda triggers for a user pool. When a trigger
 is specified with a `nil` value,
// it is removed from the user pool.
func (actor CognitoActions) UpdateTriggers(ctx context.Context, userPoolId
 string, triggers ...TriggerInfo) error {
 output, err := actor.CognitoClient.DescribeUserPool(ctx,
 &cognitoidentityprovider.DescribeUserPoolInput{
  UserPoolId: aws.String(userPoolId),
 })
 if err != nil {
  log.Printf("Couldn't get info about user pool %v. Here's why: %v\n",
 userPoolId, err)
  return err
 }
 lambdaConfig := output.UserPool.LambdaConfig
 for _, trigger := range triggers {
  switch trigger.Trigger {
  case PreSignUp:
   lambdaConfig.PreSignUp = trigger.HandlerArn
  case UserMigration:
   lambdaConfig.UserMigration = trigger.HandlerArn
  case PostAuthentication:
   lambdaConfig.PostAuthentication = trigger.HandlerArn
  }
 }
 _, err = actor.CognitoClient.UpdateUserPool(ctx,
 &cognitoidentityprovider.UpdateUserPoolInput{
  UserPoolId:   aws.String(userPoolId),
  LambdaConfig: lambdaConfig,
 })
 if err != nil {
  log.Printf("Couldn't update user pool %v. Here's why: %v\n", userPoolId, err)
 }
 return err
}



// SignUp signs up a user with Amazon Cognito.
```

```
func (actor CognitoActions) SignUp(ctx context.Context, clientId string, userName
 string, password string, userEmail string) (bool, error) {
 confirmed := false
 output, err := actor.CognitoClient.SignUp(ctx,
 &cognitoidentityprovider.SignUpInput{
  ClientId: aws.String(clientId),
  Password: aws.String(password),
  Username: aws.String(userName),
  UserAttributes: []types.AttributeType{
   {Name: aws.String("email"), Value: aws.String(userEmail)},
  },
 })
 if err != nil {
  var invalidPassword *types.InvalidPasswordException
  if errors.As(err, &invalidPassword) {
   log.Println(*invalidPassword.Message)
  } else {
   log.Printf("Couldn't sign up user %v. Here's why: %v\n", userName, err)
  }
 } else {
  confirmed = output.UserConfirmed
 }
 return confirmed, err
}



// SignIn signs in a user to Amazon Cognito using a username and password
 authentication flow.
func (actor CognitoActions) SignIn(ctx context.Context, clientId string, userName
 string, password string) (*types.AuthenticationResultType, error) {
 var authResult *types.AuthenticationResultType
 output, err := actor.CognitoClient.InitiateAuth(ctx,
 &cognitoidentityprovider.InitiateAuthInput{
  AuthFlow:       "USER_PASSWORD_AUTH",
  ClientId:       aws.String(clientId),
  AuthParameters: map[string]string{"USERNAME": userName, "PASSWORD": password},
 })
 if err != nil {
  var resetRequired *types.PasswordResetRequiredException
  if errors.As(err, &resetRequired) {
   log.Println(*resetRequired.Message)
  } else {
   log.Printf("Couldn't sign in user %v. Here's why: %v\n", userName, err)
```

```
  }
 } else {
  authResult = output.AuthenticationResult
 }
 return authResult, err
}



// ForgotPassword starts a password recovery flow for a user. This flow typically
 sends a confirmation code
// to the user's configured notification destination, such as email.
func (actor CognitoActions) ForgotPassword(ctx context.Context, clientId string,
 userName string) (*types.CodeDeliveryDetailsType, error) {
 output, err := actor.CognitoClient.ForgotPassword(ctx,
 &cognitoidentityprovider.ForgotPasswordInput{
  ClientId: aws.String(clientId),
  Username: aws.String(userName),
 })
 if err != nil {
  log.Printf("Couldn't start password reset for user '%v'. Here;s why: %v\n",
 userName, err)
 }
 return output.CodeDeliveryDetails, err
}



// ConfirmForgotPassword confirms a user with a confirmation code and a new
 password.
func (actor CognitoActions) ConfirmForgotPassword(ctx context.Context, clientId
 string, code string, userName string, password string) error {
 _, err := actor.CognitoClient.ConfirmForgotPassword(ctx,
 &cognitoidentityprovider.ConfirmForgotPasswordInput{
  ClientId:         aws.String(clientId),
  ConfirmationCode: aws.String(code),
  Password:         aws.String(password),
  Username:         aws.String(userName),
 })
 if err != nil {
  var invalidPassword *types.InvalidPasswordException
  if errors.As(err, &invalidPassword) {
   log.Println(*invalidPassword.Message)
  } else {
```

```
   log.Printf("Couldn't confirm user %v. Here's why: %v", userName, err)
  }
 }
 return err
}



// DeleteUser removes a user from the user pool.
func (actor CognitoActions) DeleteUser(ctx context.Context, userAccessToken
 string) error {
 _, err := actor.CognitoClient.DeleteUser(ctx,
 &cognitoidentityprovider.DeleteUserInput{
  AccessToken: aws.String(userAccessToken),
 })
 if err != nil {
  log.Printf("Couldn't delete user. Here's why: %v\n", err)
 }
 return err
}



// AdminCreateUser uses administrator credentials to add a user to a user pool.
 This method leaves the user
// in a state that requires they enter a new password next time they sign in.
func (actor CognitoActions) AdminCreateUser(ctx context.Context, userPoolId
 string, userName string, userEmail string) error {
 _, err := actor.CognitoClient.AdminCreateUser(ctx,
 &cognitoidentityprovider.AdminCreateUserInput{
  UserPoolId:     aws.String(userPoolId),
  Username:       aws.String(userName),
  MessageAction:  types.MessageActionTypeSuppress,
  UserAttributes: []types.AttributeType{{Name: aws.String("email"), Value:
 aws.String(userEmail)}},
 })
 if err != nil {
  var userExists *types.UsernameExistsException
  if errors.As(err, &userExists) {
   log.Printf("User %v already exists in the user pool.", userName)
   err = nil
  } else {
   log.Printf("Couldn't create user %v. Here's why: %v\n", userName, err)
  }
```

```
 }
 return err
}



// AdminSetUserPassword uses administrator credentials to set a password for a
 user without requiring a
// temporary password.
func (actor CognitoActions) AdminSetUserPassword(ctx context.Context, userPoolId
 string, userName string, password string) error {
 _, err := actor.CognitoClient.AdminSetUserPassword(ctx,
 &cognitoidentityprovider.AdminSetUserPasswordInput{
  Password:   aws.String(password),
  UserPoolId: aws.String(userPoolId),
  Username:   aws.String(userName),
  Permanent:  true,
 })
 if err != nil {
  var invalidPassword *types.InvalidPasswordException
  if errors.As(err, &invalidPassword) {
   log.Println(*invalidPassword.Message)
  } else {
   log.Printf("Couldn't set password for user %v. Here's why: %v\n", userName,
 err)
  }
 }
 return err
}
```

Buat struct yang membungkus tindakan DynamoDB.

```
import (
 "context"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
```

```
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// DynamoActions encapsulates the Amazon Simple Notification Service (Amazon SNS)
 actions
// used in the examples.
type DynamoActions struct {
 DynamoClient *dynamodb.Client
}

// User defines structured user data.
type User struct {
 UserName  string
 UserEmail string
 LastLogin *LoginInfo `dynamodbav:",omitempty"`
}

// LoginInfo defines structured custom login data.
type LoginInfo struct {
 UserPoolId string
 ClientId   string
 Time       string
}

// UserList defines a list of users.
type UserList struct {
 Users []User
}

// UserNameList returns the usernames contained in a UserList as a list of
 strings.
func (users *UserList) UserNameList() []string {
 names := make([]string, len(users.Users))
 for i := 0; i < len(users.Users); i++ {
  names[i] = users.Users[i].UserName
 }
 return names
}

// PopulateTable adds a set of test users to the table.
func (actor DynamoActions) PopulateTable(ctx context.Context, tableName string)
 error {
 var err error
 var item map[string]types.AttributeValue
```

```
 var writeReqs []types.WriteRequest
 for i := 1; i < 4; i++ {
  item, err = attributevalue.MarshalMap(User{UserName: fmt.Sprintf("test_user_
%v", i), UserEmail: fmt.Sprintf("test_email_%v@example.com", i)})
  if err != nil {
   log.Printf("Couldn't marshall user into DynamoDB format. Here's why: %v\n",
 err)
   return err
  }
  writeReqs = append(writeReqs, types.WriteRequest{PutRequest:
 &types.PutRequest{Item: item}})
 }
 _, err = actor.DynamoClient.BatchWriteItem(ctx, &dynamodb.BatchWriteItemInput{
  RequestItems: map[string][]types.WriteRequest{tableName: writeReqs},
 })
 if err != nil {
  log.Printf("Couldn't populate table %v with users. Here's why: %v\n",
 tableName, err)
 }
 return err
}

// Scan scans the table for all items.
func (actor DynamoActions) Scan(ctx context.Context, tableName string) (UserList,
 error) {
 var userList UserList
 output, err := actor.DynamoClient.Scan(ctx, &dynamodb.ScanInput{
  TableName: aws.String(tableName),
 })
 if err != nil {
  log.Printf("Couldn't scan table %v for items. Here's why: %v\n", tableName,
 err)
 } else {
  err = attributevalue.UnmarshalListOfMaps(output.Items, &userList.Users)
  if err != nil {
   log.Printf("Couldn't unmarshal items into users. Here's why: %v\n", err)
  }
 }
 return userList, err
}

// AddUser adds a user item to a table.
func (actor DynamoActions) AddUser(ctx context.Context, tableName string, user
 User) error {
```

```
 userItem, err := attributevalue.MarshalMap(user)
 if err != nil {
  log.Printf("Couldn't marshall user to item. Here's why: %v\n", err)
 }
 _, err = actor.DynamoClient.PutItem(ctx, &dynamodb.PutItemInput{
  Item:      userItem,
  TableName: aws.String(tableName),
 })
 if err != nil {
  log.Printf("Couldn't put item in table %v. Here's why: %v", tableName, err)
 }
 return err
}
```

Buat struct yang membungkus CloudWatch Tindakan log.

```
import (
 "context"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs"
 "github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs/types"
)

type CloudWatchLogsActions struct {
 CwlClient *cloudwatchlogs.Client
}

// GetLatestLogStream gets the most recent log stream for a Lambda function.
func (actor CloudWatchLogsActions) GetLatestLogStream(ctx context.Context,
 functionName string) (types.LogStream, error) {
 var logStream types.LogStream
 logGroupName := fmt.Sprintf("/aws/lambda/%s", functionName)
 output, err := actor.CwlClient.DescribeLogStreams(ctx,
 &cloudwatchlogs.DescribeLogStreamsInput{
  Descending:   aws.Bool(true),
  Limit:        aws.Int32(1),
  LogGroupName: aws.String(logGroupName),
```

```
  OrderBy:      types.OrderByLastEventTime,
 })
 if err != nil {
  log.Printf("Couldn't get log streams for log group %v. Here's why: %v\n",
 logGroupName, err)
 } else {
  logStream = output.LogStreams[0]
 }
 return logStream, err
}

// GetLogEvents gets the most recent eventCount events from the specified log
 stream.
func (actor CloudWatchLogsActions) GetLogEvents(ctx context.Context, functionName
 string, logStreamName string, eventCount int32) (
 []types.OutputLogEvent, error) {
 var events []types.OutputLogEvent
 logGroupName := fmt.Sprintf("/aws/lambda/%s", functionName)
 output, err := actor.CwlClient.GetLogEvents(ctx,
 &cloudwatchlogs.GetLogEventsInput{
  LogStreamName: aws.String(logStreamName),
  Limit:         aws.Int32(eventCount),
  LogGroupName:  aws.String(logGroupName),
 })
 if err != nil {
  log.Printf("Couldn't get log event for log stream %v. Here's why: %v\n",
 logStreamName, err)
 } else {
  events = output.Events
 }
 return events, err
}
```

Buat struct yang membungkus CloudFormation tindakan.

```
import (
 "context"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
```

```
 "github.com/aws/aws-sdk-go-v2/service/cloudformation"
)

// StackOutputs defines a map of outputs from a specific stack.
type StackOutputs map[string]string

type CloudFormationActions struct {
 CfnClient *cloudformation.Client
}

// GetOutputs gets the outputs from a CloudFormation stack and puts them into a
 structured format.
func (actor CloudFormationActions) GetOutputs(ctx context.Context, stackName
 string) StackOutputs {
 output, err := actor.CfnClient.DescribeStacks(ctx,
 &cloudformation.DescribeStacksInput{
  StackName: aws.String(stackName),
 })
 if err != nil || len(output.Stacks) == 0 {
  log.Panicf("Couldn't find a CloudFormation stack named %v. Here's why: %v\n",
 stackName, err)
 }
 stackOutputs := StackOutputs{}
 for _, out := range output.Stacks[0].Outputs {
  stackOutputs[*out.OutputKey] = *out.OutputValue
 }
 return stackOutputs
}
```

Pembersihan sumber daya

```
import (
 "context"
 "log"
 "user_pools_and_lambda_triggers/actions"

 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

// Resources keeps track of AWS resources created during an example and handles
```

```
// cleanup when the example finishes.
type Resources struct {
 userPoolId       string
 userAccessTokens []string
 triggers         []actions.Trigger

 cognitoActor *actions.CognitoActions
 questioner   demotools.IQuestioner
}

func (resources *Resources) init(cognitoActor *actions.CognitoActions, questioner
 demotools.IQuestioner) {
 resources.userAccessTokens = []string{}
 resources.triggers = []actions.Trigger{}
 resources.cognitoActor = cognitoActor
 resources.questioner = questioner
}

// Cleanup deletes all AWS resources created during an example.
func (resources *Resources) Cleanup(ctx context.Context) {
 defer func() {
  if r := recover(); r != nil {
   log.Printf("Something went wrong during cleanup.\n%v\n", r)
   log.Println("Use the AWS Management Console to remove any remaining resources
 \n" +
    "that were created for this scenario.")
  }
 }()

 wantDelete := resources.questioner.AskBool("Do you want to remove all of the AWS
 resources that were created "+
  "during this demo (y/n)?", "y")
 if wantDelete {
  for _, accessToken := range resources.userAccessTokens {
   err := resources.cognitoActor.DeleteUser(ctx, accessToken)
   if err != nil {
    log.Println("Couldn't delete user during cleanup.")
    panic(err)
   }
   log.Println("Deleted user.")
  }
  triggerList := make([]actions.TriggerInfo, len(resources.triggers))
  for i := 0; i < len(resources.triggers); i++ {
```

```
    triggerList[i] = actions.TriggerInfo{Trigger: resources.triggers[i],
  HandlerArn: nil}
   }
   err := resources.cognitoActor.UpdateTriggers(ctx, resources.userPoolId,
  triggerList...)
   if err != nil {
    log.Println("Couldn't update Cognito triggers during cleanup.")
    panic(err)
   }
   log.Println("Removed Cognito triggers from user pool.")
  } else {
   log.Println("Be sure to remove resources when you're done with them to avoid
  unexpected charges!")
  }
 }
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK untuk Go .
  - AdminCreateUser
  - AdminSetUserPassword
  - DeleteUser
  - InitiateAuth
  - UpdateUserPool

Untuk daftar lengkap AWS Panduan pengembang SDK dan contoh kode, lihat Menggunakan Lambda dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Contoh tanpa server untuk Lambda

Contoh kode berikut menunjukkan cara menggunakan Lambda dengan AWS SDK.

Contoh

- Menghubungkan ke database Amazon RDS dalam fungsi Lambda
- Memanggil fungsi Lambda dari pemicu Kinesis
- Memanggil fungsi Lambda dari pemicu DynamoDB
- Memanggil fungsi Lambda dari pemicu Amazon DocumentDB

- Memanggil fungsi Lambda dari pemicu Amazon MSK
- Menginvokasi fungsi Lambda dari pemicu Amazon S3
- Memanggil fungsi Lambda dari pemicu Amazon SNS
- Memanggil fungsi Lambda dari pemicu Amazon SQS
- Melaporkan kegagalan item batch untuk fungsi Lambda dengan pemicu Kinesis
- Melaporkan kegagalan item batch untuk fungsi Lambda dengan pemicu DynamoDB
- Melaporkan kegagalan item batch untuk fungsi Lambda dengan pemicu Amazon SQS

Menghubungkan ke database Amazon RDS dalam fungsi Lambda

Contoh kode berikut menunjukkan bagaimana menerapkan fungsi Lambda yang terhubung ke database RDS. Fungsi membuat permintaan database sederhana dan mengembalikan hasilnya.

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Menghubungkan ke database Amazon RDS dalam fungsi Lambda menggunakan.NET.

```
using System.Data;
using System.Text.Json;
using Amazon.Lambda.APIGatewayEvents;
using Amazon.Lambda.Core;
using MySql.Data.MySqlClient;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri

namespace aws_rds;

public class InputModel
```

```
{
    public string key1 { get; set; }
    public string key2 { get; set; }
}

public class Function
{
    /// <summary>
    // Handles the Lambda function execution for connecting to RDS using IAM
 authentication.
    /// </summary>
    /// <param name="input">The input event data passed to the Lambda function</
param>
    /// <param name="context">The Lambda execution context that provides runtime
 information</param>
    /// <returns>A response object containing the execution result</returns>

    public async Task<APIGatewayProxyResponse>
 FunctionHandler(APIGatewayProxyRequest request, ILambdaContext context)
    {
        // Sample Input: {"body": "{\"key1\":\"20\", \"key2\":\"25\"}"}
        var input = JsonSerializer.Deserialize<InputModel>(request.Body);

        /// Obtain authentication token
        var authToken = RDSAuthTokenGenerator.GenerateAuthToken(
            Environment.GetEnvironmentVariable("RDS_ENDPOINT"),
            Convert.ToInt32(Environment.GetEnvironmentVariable("RDS_PORT")),
            Environment.GetEnvironmentVariable("RDS_USERNAME")
        );

        /// Build the Connection String with the Token
        string connectionString =
 $"Server={Environment.GetEnvironmentVariable("RDS_ENDPOINT")};" +

 $"Port={Environment.GetEnvironmentVariable("RDS_PORT")};" +

 $"Uid={Environment.GetEnvironmentVariable("RDS_USERNAME")};" +
                                  $"Pwd={authToken};";


        try
        {
            await using var connection = new MySqlConnection(connectionString);
            await connection.OpenAsync();
```

```
            const string sql = "SELECT @param1 + @param2 AS Sum";

            await using var command = new MySqlCommand(sql, connection);
            command.Parameters.AddWithValue("@param1", int.Parse(input.key1 ??
 "0"));
            command.Parameters.AddWithValue("@param2", int.Parse(input.key2 ??
 "0"));

            await using var reader = await command.ExecuteReaderAsync();
            if (await reader.ReadAsync())
            {
                int result = reader.GetInt32("Sum");

                //Sample Response: {"statusCode":200,"body":"{\"message\":\"The
 sum is: 45\"}","isBase64Encoded":false}
                return new APIGatewayProxyResponse
                {
                    StatusCode = 200,
                    Body = JsonSerializer.Serialize(new { message = $"The sum is:
 {result}" })
                };
            }

        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error: {ex.Message}");
        }

        return new APIGatewayProxyResponse
        {
            StatusCode = 500,
            Body = JsonSerializer.Serialize(new { error = "Internal server
 error" })
        };
    }
}
```

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Menghubungkan ke database Amazon RDS dalam fungsi Lambda menggunakan Go.

```
/*
Golang v2 code here.
*/

package main

import (
 "context"
 "database/sql"
 "encoding/json"
 "fmt"
 "os"

 "github.com/aws/aws-lambda-go/lambda"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/feature/rds/auth"
 _ "github.com/go-sql-driver/mysql"
)

type MyEvent struct {
 Name string `json:"name"`
}

func HandleRequest(event *MyEvent) (map[string]interface{}, error) {

 var dbName string = os.Getenv("DatabaseName")
 var dbUser string = os.Getenv("DatabaseUser")
 var dbHost string = os.Getenv("DBHost") // Add hostname without https
 var dbPort int = os.Getenv("Port")      // Add port number
 var dbEndpoint string = fmt.Sprintf("%s:%d", dbHost, dbPort)
 var region string = os.Getenv("AWS_REGION")
```

```
 cfg, err := config.LoadDefaultConfig(context.TODO())
 if err != nil {
  panic("configuration error: " + err.Error())
 }

 authenticationToken, err := auth.BuildAuthToken(
  context.TODO(), dbEndpoint, region, dbUser, cfg.Credentials)
 if err != nil {
  panic("failed to create authentication token: " + err.Error())
 }

 dsn := fmt.Sprintf("%s:%s@tcp(%s)/%s?tls=true&allowCleartextPasswords=true",
  dbUser, authenticationToken, dbEndpoint, dbName,
 )

 db, err := sql.Open("mysql", dsn)
 if err != nil {
  panic(err)
 }

 defer db.Close()

 var sum int
 err = db.QueryRow("SELECT ?+? AS sum", 3, 2).Scan(&sum)
 if err != nil {
  panic(err)
 }
 s := fmt.Sprint(sum)
 message := fmt.Sprintf("The selected sum is: %s", s)

 messageBytes, err := json.Marshal(message)
 if err != nil {
  return nil, err
 }

 messageString := string(messageBytes)
 return map[string]interface{}{
  "statusCode": 200,
  "headers":    map[string]string{"Content-Type": "application/json"},
  "body":       messageString,
 }, nil
}
```

```
func main() {
 lambda.Start(HandleRequest)
}
```

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Menghubungkan ke database Amazon RDS dalam fungsi Lambda menggunakan Java.

```
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.APIGatewayProxyRequestEvent;
import com.amazonaws.services.lambda.runtime.events.APIGatewayProxyResponseEvent;
import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rdsdata.RdsDataClient;
import software.amazon.awssdk.services.rdsdata.model.ExecuteStatementRequest;
import software.amazon.awssdk.services.rdsdata.model.ExecuteStatementResponse;
import software.amazon.awssdk.services.rdsdata.model.Field;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;

public class RdsLambdaHandler implements
 RequestHandler<APIGatewayProxyRequestEvent, APIGatewayProxyResponseEvent> {

    @Override
    public APIGatewayProxyResponseEvent handleRequest(APIGatewayProxyRequestEvent
 event, Context context) {
        APIGatewayProxyResponseEvent response = new
 APIGatewayProxyResponseEvent();
```

```
        try {
            // Obtain auth token
            String token = createAuthToken();

            // Define connection configuration
            String connectionString = String.format("jdbc:mysql://%s:%s/%s?
useSSL=true&requireSSL=true",
                    System.getenv("ProxyHostName"),
                    System.getenv("Port"),
                    System.getenv("DBName"));

            // Establish a connection to the database
            try (Connection connection =
 DriverManager.getConnection(connectionString, System.getenv("DBUserName"),
 token);
                 PreparedStatement statement =
 connection.prepareStatement("SELECT ? + ? AS sum")) {

                statement.setInt(1, 3);
                statement.setInt(2, 2);

                try (ResultSet resultSet = statement.executeQuery()) {
                    if (resultSet.next()) {
                        int sum = resultSet.getInt("sum");
                        response.setStatusCode(200);
                        response.setBody("The selected sum is: " + sum);
                    }
                }
            }

        } catch (Exception e) {
            response.setStatusCode(500);
            response.setBody("Error: " + e.getMessage());
        }

        return response;
    }

    private String createAuthToken() {
        // Create RDS Data Service client
        RdsDataClient rdsDataClient = RdsDataClient.builder()
                .region(Region.of(System.getenv("AWS_REGION")))
                .credentialsProvider(DefaultCredentialsProvider.create())
                .build();
```

```
        // Define authentication request
        ExecuteStatementRequest request = ExecuteStatementRequest.builder()
                .resourceArn(System.getenv("ProxyHostName"))
                .secretArn(System.getenv("DBUserName"))
                .database(System.getenv("DBName"))
                .sql("SELECT 'RDS IAM Authentication'")
                .build();

        // Execute request and obtain authentication token
        ExecuteStatementResponse response =
 rdsDataClient.executeStatement(request);
        Field tokenField = response.records().get(0).get(0);

        return tokenField.stringValue();
    }
}
```

JavaScript

SDK untuk JavaScript (V3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Menghubungkan ke database Amazon RDS dalam fungsi Lambda menggunakan JavaScript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
/*
Node.js code here.
*/
// ES6+ example
import { Signer } from "@aws-sdk/rds-signer";
import mysql from 'mysql2/promise';

async function createAuthToken() {
  // Define connection authentication parameters
```

```
  const dbinfo = {

    hostname: process.env.ProxyHostName,
    port: process.env.Port,
    username: process.env.DBUserName,
    region: process.env.AWS_REGION,

  }

  // Create RDS Signer object
  const signer = new Signer(dbinfo);

  // Request authorization token from RDS, specifying the username
  const token = await signer.getAuthToken();
  return token;
}

async function dbOps() {

  // Obtain auth token
  const token = await createAuthToken();
  // Define connection configuration
  let connectionConfig = {
    host: process.env.ProxyHostName,
    user: process.env.DBUserName,
    password: token,
    database: process.env.DBName,
    ssl: 'Amazon RDS'
  }
  // Create the connection to the DB
  const conn = await mysql.createConnection(connectionConfig);
  // Obtain the result of the query
  const [res,] = await conn.execute('select ?+? as sum', [3, 2]);
  return res;

}

export const handler = async (event) => {
  // Execute database flow
  const result = await dbOps();
  // Return result
  return {
    statusCode: 200,
    body: JSON.stringify("The selected sum is: " + result[0].sum)
```

```
  }
};
```

Menghubungkan ke database Amazon RDS dalam fungsi Lambda menggunakan TypeScript.

```
import { Signer } from "@aws-sdk/rds-signer";
import mysql from 'mysql2/promise';

// RDS settings
// Using '!' (non-null assertion operator) to tell the TypeScript compiler that
 the DB settings are not null or undefined,
const proxy_host_name = process.env.PROXY_HOST_NAME!
const port = parseInt(process.env.PORT!)
const db_name = process.env.DB_NAME!
const db_user_name = process.env.DB_USER_NAME!
const aws_region = process.env.AWS_REGION!


async function createAuthToken(): Promise<string> {

    // Create RDS Signer object
    const signer = new Signer({
        hostname: proxy_host_name,
        port: port,
        region: aws_region,
        username: db_user_name
    });

    // Request authorization token from RDS, specifying the username
    const token = await signer.getAuthToken();
    return token;
}

async function dbOps(): Promise<mysql.QueryResult | undefined> {
    try {
        // Obtain auth token
        const token = await createAuthToken();
        const conn = await mysql.createConnection({
            host: proxy_host_name,
            user: db_user_name,
            password: token,
```

```
            database: db_name,
            ssl: 'Amazon RDS' // Ensure you have the CA bundle for SSL connection
        });
        const [rows, fields] = await conn.execute('SELECT ? + ? AS sum', [3, 2]);
        console.log('result:', rows);
        return rows;
    }
    catch (err) {
        console.log(err);
    }
}

export const lambdaHandler = async (event: any): Promise<{ statusCode: number;
 body: string }> => {
    // Execute database flow
    const result = await dbOps();

    // Return error is result is undefined
    if (result == undefined)
        return {
            statusCode: 500,
            body: JSON.stringify(`Error with connection to DB host`)
        }

    // Return result
    return {
        statusCode: 200,
        body: JSON.stringify(`The selected sum is: ${result[0].sum}`)
    };
};
```

PHP

SDK untuk PHP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Menghubungkan ke database Amazon RDS dalam fungsi Lambda menggunakan PHP.

```
<?php
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\Handler as StdHandler;
use Bref\Logger\StderrLogger;
use Aws\Rds\AuthTokenGenerator;
use Aws\Credentials\CredentialProvider;

require __DIR__ . '/vendor/autoload.php';

class Handler implements StdHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }


    private function getAuthToken(): string {
        // Define connection authentication parameters
        $dbConnection = [
            'hostname' => getenv('DB_HOSTNAME'),
            'port' => getenv('DB_PORT'),
            'username' => getenv('DB_USERNAME'),
            'region' => getenv('AWS_REGION'),
        ];

        // Create RDS AuthTokenGenerator object
        $generator = new
 AuthTokenGenerator(CredentialProvider::defaultProvider());

        // Request authorization token from RDS, specifying the username
        return $generator->createToken(
            $dbConnection['hostname'] . ':' . $dbConnection['port'],
            $dbConnection['region'],
            $dbConnection['username']
        );
    }
```

```
    private function getQueryResults() {
        // Obtain auth token
        $token = $this->getAuthToken();

        // Define connection configuration
        $connectionConfig = [
            'host' => getenv('DB_HOSTNAME'),
            'user' => getenv('DB_USERNAME'),
            'password' => $token,
            'database' => getenv('DB_NAME'),
        ];

        // Create the connection to the DB
        $conn = new PDO(

"mysql:host={$connectionConfig['host']};dbname={$connectionConfig['database']}",
            $connectionConfig['user'],
            $connectionConfig['password'],
            [
                PDO::MYSQL_ATTR_SSL_CA => '/path/to/rds-ca-2019-root.pem',
                PDO::MYSQL_ATTR_SSL_VERIFY_SERVER_CERT => true,
            ]
        );

        // Obtain the result of the query
        $stmt = $conn->prepare('SELECT ?+? AS sum');
        $stmt->execute([3, 2]);

        return $stmt->fetch(PDO::FETCH_ASSOC);
    }

    /**
     * @param mixed $event
     * @param Context $context
     * @return array
     */
    public function handle(mixed $event, Context $context): array
    {
        $this->logger->info("Processing query");

        // Execute database flow
        $result = $this->getQueryResults();
```

```
        return [
            'sum' => $result['sum']
        ];
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Menghubungkan ke database Amazon RDS dalam fungsi Lambda menggunakan Python.

```
import json
import os
import boto3
import pymysql

# RDS settings
proxy_host_name = os.environ['PROXY_HOST_NAME']
port = int(os.environ['PORT'])
db_name = os.environ['DB_NAME']
db_user_name = os.environ['DB_USER_NAME']
aws_region = os.environ['AWS_REGION']


# Fetch RDS Auth Token
def get_auth_token():
    client = boto3.client('rds')
    token = client.generate_db_auth_token(
        DBHostname=proxy_host_name,
        Port=port
        DBUsername=db_user_name
        Region=aws_region
```

```
    )
    return token

def lambda_handler(event, context):
    token = get_auth_token()
    try:
        connection = pymysql.connect(
            host=proxy_host_name,
            user=db_user_name,
            password=token,
            db=db_name,
            port=port,
            ssl={'ca': 'Amazon RDS'}  # Ensure you have the CA bundle for SSL
 connection
        )

        with connection.cursor() as cursor:
            cursor.execute('SELECT %s + %s AS sum', (3, 2))
            result = cursor.fetchone()

        return result

    except Exception as e:
        return (f"Error: {str(e)}")  # Return an error message if an exception
 occurs
```

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Menghubungkan ke database Amazon RDS dalam fungsi Lambda menggunakan Ruby.

```
# Ruby code here.

require 'aws-sdk-rds'
```

```
require 'json'
require 'mysql2'

def lambda_handler(event:, context:)
  endpoint = ENV['DBEndpoint'] # Add the endpoint without https"
  port = ENV['Port']           # 3306
  user = ENV['DBUser']
  region = ENV['DBRegion']     # 'us-east-1'
  db_name = ENV['DBName']

  credentials = Aws::Credentials.new(
    ENV['AWS_ACCESS_KEY_ID'],
    ENV['AWS_SECRET_ACCESS_KEY'],
    ENV['AWS_SESSION_TOKEN']
  )
  rds_client = Aws::RDS::AuthTokenGenerator.new(
    region: region,
    credentials: credentials
  )

  token = rds_client.auth_token(
    endpoint: endpoint+ ':' + port,
    user_name: user,
    region: region
  )

  begin
    conn = Mysql2::Client.new(
      host: endpoint,
      username: user,
      password: token,
      port: port,
      database: db_name,
      sslca: '/var/task/global-bundle.pem',
      sslverify: true,
      enable_cleartext_plugin: true
    )
    a = 3
    b = 2
    result = conn.query("SELECT #{a} + #{b} AS sum").first['sum']
    puts result
    conn.close
    {
      statusCode: 200,
```

```
      body: result.to_json
    }
  rescue => e
    puts "Database connection failed due to #{e}"
  end
end
```

Rust

SDK for Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Menghubungkan ke database Amazon RDS dalam fungsi Lambda menggunakan Rust.

```
use aws_config::BehaviorVersion;
use aws_credential_types::provider::ProvideCredentials;
use aws_sigv4::{
    http_request::{sign, SignableBody, SignableRequest, SigningSettings},
    sign::v4,
};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};
use serde_json::{json, Value};
use sqlx::postgres::PgConnectOptions;
use std::env;
use std::time::{Duration, SystemTime};

const RDS_CERTS: &[u8] = include_bytes!("global-bundle.pem");

async fn generate_rds_iam_token(
    db_hostname: &str,
    port: u16,
    db_username: &str,
) -> Result<String, Error> {
    let config = aws_config::load_defaults(BehaviorVersion::v2024_03_28()).await;

    let credentials = config
        .credentials_provider()
```

```
        .expect("no credentials provider found")
        .provide_credentials()
        .await
        .expect("unable to load credentials");
    let identity = credentials.into();
    let region = config.region().unwrap().to_string();

    let mut signing_settings = SigningSettings::default();
    signing_settings.expires_in = Some(Duration::from_secs(900));
    signing_settings.signature_location =
 aws_sigv4::http_request::SignatureLocation::QueryParams;

    let signing_params = v4::SigningParams::builder()
        .identity(&identity)
        .region(&region)
        .name("rds-db")
        .time(SystemTime::now())
        .settings(signing_settings)
        .build()?;

    let url = format!(
        "https://{db_hostname}:{port}/?Action=connect&DBUser={db_user}",
        db_hostname = db_hostname,
        port = port,
        db_user = db_username
    );

    let signable_request =
        SignableRequest::new("GET", &url, std::iter::empty(),
 SignableBody::Bytes(&[]))
            .expect("signable request");

    let (signing_instructions, _signature) =
        sign(signable_request, &signing_params.into())?.into_parts();

    let mut url = url::Url::parse(&url).unwrap();
    for (name, value) in signing_instructions.params() {
        url.query_pairs_mut().append_pair(name, &value);
    }

    let response = url.to_string().split_off("https://".len());

    Ok(response)
}
```

```
#[tokio::main]
async fn main() -> Result<(), Error> {
    run(service_fn(handler)).await
}

async fn handler(_event: LambdaEvent<Value>) -> Result<Value, Error> {
    let db_host = env::var("DB_HOSTNAME").expect("DB_HOSTNAME must be set");
    let db_port = env::var("DB_PORT")
        .expect("DB_PORT must be set")
        .parse::<u16>()
        .expect("PORT must be a valid number");
    let db_name = env::var("DB_NAME").expect("DB_NAME must be set");
    let db_user_name = env::var("DB_USERNAME").expect("DB_USERNAME must be set");

    let token = generate_rds_iam_token(&db_host, db_port, &db_user_name).await?;

    let opts = PgConnectOptions::new()
        .host(&db_host)
        .port(db_port)
        .username(&db_user_name)
        .password(&token)
        .database(&db_name)
        .ssl_root_cert_from_pem(RDS_CERTS.to_vec())
        .ssl_mode(sqlx::postgres::PgSslMode::Require);

    let pool = sqlx::postgres::PgPoolOptions::new()
        .connect_with(opts)
        .await?;

    let result: i32 = sqlx::query_scalar("SELECT $1 + $2")
        .bind(3)
        .bind(2)
        .fetch_one(&pool)
        .await?;

    println!("Result: {:?}", result);

    Ok(json!({
        "statusCode": 200,
        "content-type": "text/plain",
        "body": format!("The selected sum is: {result}")
    }))
}
```

Untuk daftar lengkap AWS Panduan pengembang SDK dan contoh kode, lihat Menggunakan Lambda dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Memanggil fungsi Lambda dari pemicu Kinesis

Contoh kode berikut menunjukkan bagaimana menerapkan fungsi Lambda yang menerima peristiwa yang dipicu oleh menerima catatan dari aliran Kinesis. Fungsi ini mengambil payload Kinesis, mendekode dari Base64, dan mencatat konten rekaman.

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara Kinesis dengan Lambda menggunakan.NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.KinesisEvents;
using AWS.Lambda.Powertools.Logging;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri

namespace KinesisIntegrationSampleCode;

public class Function
{
    // Powertools Logger requires an environment variables against your function
```

```
    // POWERTOOLS_SERVICE_NAME
    [Logging(LogEvent = true)]
    public async Task FunctionHandler(KinesisEvent evnt, ILambdaContext context)
    {
        if (evnt.Records.Count == 0)
        {
            Logger.LogInformation("Empty Kinesis Event received");
            return;
        }

        foreach (var record in evnt.Records)
        {
            try
            {
                Logger.LogInformation($"Processed Event with EventId:
 {record.EventId}");
                string data = await GetRecordDataAsync(record.Kinesis, context);
                Logger.LogInformation($"Data: {data}");
                // TODO: Do interesting work based on the new data
            }
            catch (Exception ex)
            {
                Logger.LogError($"An error occurred {ex.Message}");
                throw;
            }
        }
        Logger.LogInformation($"Successfully processed {evnt.Records.Count}
 records.");
    }

    private async Task<string> GetRecordDataAsync(KinesisEvent.Record record,
 ILambdaContext context)
    {
        byte[] bytes = record.Data.ToArray();
        string data = Encoding.UTF8.GetString(bytes);
        await Task.CompletedTask; //Placeholder for actual async work
        return data;
    }
}
```

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara Kinesis dengan Lambda menggunakan Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "log"

 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

func handler(ctx context.Context, kinesisEvent events.KinesisEvent) error {
 if len(kinesisEvent.Records) == 0 {
  log.Printf("empty Kinesis event received")
  return nil
 }

 for _, record := range kinesisEvent.Records {
  log.Printf("processed Kinesis event with EventId: %v", record.EventID)
  recordDataBytes := record.Kinesis.Data
  recordDataText := string(recordDataBytes)
  log.Printf("record data: %v", recordDataText)
  // TODO: Do interesting work based on the new data
 }
 log.Printf("successfully processed %v records", len(kinesisEvent.Records))
 return nil
}

func main() {
 lambda.Start(handler)
```

```
}
```

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara Kinesis dengan Lambda menggunakan Java.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.LambdaLogger;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.KinesisEvent;

public class Handler implements RequestHandler<KinesisEvent, Void> {
    @Override
    public Void handleRequest(final KinesisEvent event, final Context context) {
        LambdaLogger logger = context.getLogger();
        if (event.getRecords().isEmpty()) {
            logger.log("Empty Kinesis Event received");
            return null;
        }
        for (KinesisEvent.KinesisEventRecord record : event.getRecords()) {
            try {
                logger.log("Processed Event with EventId: "+record.getEventID());
                String data = new String(record.getKinesis().getData().array());
                logger.log("Data:"+ data);
                // TODO: Do interesting work based on the new data
            }
            catch (Exception ex) {
                logger.log("An error occurred:"+ex.getMessage());
                throw ex;
```

```
            }
        }
        logger.log("Successfully processed:"+event.getRecords().size()+"
 records");
        return null;
    }

}
```

JavaScript

SDK untuk JavaScript (V3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara Kinesis dengan menggunakan Lambda JavaScript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const record of event.Records) {
    try {
      console.log(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      console.log(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      console.error(`An error occurred ${err}`);
      throw err;
    }
  }
  console.log(`Successfully processed ${event.Records.length} records.`);
};

async function getRecordDataAsync(payload) {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
```

```
  return data;
}
```

Mengkonsumsi acara Kinesis dengan menggunakan Lambda TypeScript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  KinesisStreamEvent,
  Context,
  KinesisStreamHandler,
  KinesisStreamRecordPayload,
} from "aws-lambda";
import { Buffer } from "buffer";
import { Logger } from "@aws-lambda-powertools/logger";

const logger = new Logger({
  logLevel: "INFO",
  serviceName: "kinesis-stream-handler-sample",
});

export const functionHandler: KinesisStreamHandler = async (
  event: KinesisStreamEvent,
  context: Context
): Promise<void> => {
  for (const record of event.Records) {
    try {
      logger.info(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      logger.info(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      logger.error(`An error occurred ${err}`);
      throw err;
    }
    logger.info(`Successfully processed ${event.Records.length} records.`);
  }
};

async function getRecordDataAsync(
  payload: KinesisStreamRecordPayload
): Promise<string> {
```

```
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

PHP

SDK untuk PHP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara Kinesis dengan Lambda menggunakan PHP.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\Kinesis\KinesisEvent;
use Bref\Event\Kinesis\KinesisHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler extends KinesisHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
```

```
     */
    public function handleKinesis(KinesisEvent $event, Context $context): void
    {
        $this->logger->info("Processing records");
        $records = $event->getRecords();
        foreach ($records as $record) {
            $data = $record->getData();
            $this->logger->info(json_encode($data));
            // TODO: Do interesting work based on the new data

            // Any exception thrown will be logged and the invocation will be
 marked as failed
        }
        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords records");
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara Kinesis dengan Lambda menggunakan Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
import base64
def lambda_handler(event, context):

    for record in event['Records']:
        try:
            print(f"Processed Kinesis Event - EventID: {record['eventID']}")
```

```
            record_data = base64.b64decode(record['kinesis']
['data']).decode('utf-8')
            print(f"Record Data: {record_data}")
            # TODO: Do interesting work based on the new data
        except Exception as e:
            print(f"An error occurred {e}")
            raise e
    print(f"Successfully processed {len(event['Records'])} records.")
```

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara Kinesis dengan Lambda menggunakan Ruby.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
require 'aws-sdk'

def lambda_handler(event:, context:)
  event['Records'].each do |record|
    begin
      puts "Processed Kinesis Event - EventID: #{record['eventID']}"
      record_data = get_record_data_async(record['kinesis'])
      puts "Record Data: #{record_data}"
      # TODO: Do interesting work based on the new data
    rescue => err
      $stderr.puts "An error occurred #{err}"
      raise err
    end
  end
  puts "Successfully processed #{event['Records'].length} records."
end

def get_record_data_async(payload)
```

```
  data = Base64.decode64(payload['data']).force_encoding('UTF-8')
  # Placeholder for actual async work
  # You can use Ruby's asynchronous programming tools like async/await or fibers
 here.
  return data
end
```

Rust

SDK for Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara Kinesis dengan Lambda menggunakan Rust.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::event::kinesis::KinesisEvent;
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

async fn function_handler(event: LambdaEvent<KinesisEvent>) -> Result<(), Error>
 {
    if event.payload.records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(());
    }

    event.payload.records.iter().for_each(|record| {
        tracing::info!("EventId:
 {}",record.event_id.as_deref().unwrap_or_default());

        let record_data = std::str::from_utf8(&record.kinesis.data);

        match record_data {
            Ok(data) => {
                // log the record data
                tracing::info!("Data: {}", data);
            }
```

```
            Err(e) => {
                tracing::error!("Error: {}", e);
            }
        }
    });

    tracing::info!(
        "Successfully processed {} records",
        event.payload.records.len()
    );

    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        // disable printing the name of the module in every log line.
        .with_target(false)
        // disabling time is handy because CloudWatch will add the ingestion
 time.
        .without_time()
        .init();

    run(service_fn(function_handler)).await
}
```

Untuk daftar lengkap AWS Panduan pengembang SDK dan contoh kode, lihat Menggunakan Lambda dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Memanggil fungsi Lambda dari pemicu DynamoDB

Contoh kode berikut menunjukkan bagaimana menerapkan fungsi Lambda yang menerima peristiwa yang dipicu dengan menerima catatan dari aliran DynamoDB. Fungsi mengambil payload DynamoDB dan mencatat konten rekaman.

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi peristiwa DynamoDB dengan Lambda menggunakan.NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text.Json;
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.DynamoDBEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri

namespace AWSLambda_DDB;

public class Function
{
    public void FunctionHandler(DynamoDBEvent dynamoEvent, ILambdaContext
 context)
    {
        context.Logger.LogInformation($"Beginning to process
 {dynamoEvent.Records.Count} records...");

        foreach (var record in dynamoEvent.Records)
        {
            context.Logger.LogInformation($"Event ID: {record.EventID}");
            context.Logger.LogInformation($"Event Name: {record.EventName}");

            context.Logger.LogInformation(JsonSerializer.Serialize(record));
        }

        context.Logger.LogInformation("Stream processing complete.");
```

```
    }
}
```

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi peristiwa DynamoDB dengan Lambda menggunakan Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "github.com/aws/aws-lambda-go/lambda"
 "github.com/aws/aws-lambda-go/events"
 "fmt"
)

func HandleRequest(ctx context.Context, event events.DynamoDBEvent) (*string,
 error) {
 if len(event.Records) == 0 {
  return nil, fmt.Errorf("received empty event")
 }

 for _, record := range event.Records {
   LogDynamoDBRecord(record)
 }

 message := fmt.Sprintf("Records processed: %d", len(event.Records))
 return &message, nil
}

func main() {
```

```
 lambda.Start(HandleRequest)
}

func LogDynamoDBRecord(record events.DynamoDBEventRecord){
 fmt.Println(record.EventID)
 fmt.Println(record.EventName)
 fmt.Printf("%+v\n", record.Change)
}
```

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi peristiwa DynamoDB dengan Lambda menggunakan Java.

```
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.DynamodbEvent;
import
 com.amazonaws.services.lambda.runtime.events.DynamodbEvent.DynamodbStreamRecord;
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;

public class example implements RequestHandler<DynamodbEvent, Void> {

    private static final Gson GSON = new
 GsonBuilder().setPrettyPrinting().create();

    @Override
    public Void handleRequest(DynamodbEvent event, Context context) {
        System.out.println(GSON.toJson(event));
        event.getRecords().forEach(this::logDynamoDBRecord);
        return null;
    }

    private void logDynamoDBRecord(DynamodbStreamRecord record) {
```

```
        System.out.println(record.getEventID());
        System.out.println(record.getEventName());
        System.out.println("DynamoDB Record: " +
 GSON.toJson(record.getDynamodb()));
    }
}
```

JavaScript

SDK untuk JavaScript (V3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi peristiwa DynamoDB dengan menggunakan Lambda JavaScript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
    console.log(JSON.stringify(event, null, 2));
    event.Records.forEach(record => {
        logDynamoDBRecord(record);
    });
};

const logDynamoDBRecord = (record) => {
    console.log(record.eventID);
    console.log(record.eventName);
    console.log(`DynamoDB Record: ${JSON.stringify(record.dynamodb)}`);
};
```

Mengkonsumsi peristiwa DynamoDB dengan menggunakan Lambda TypeScript.

```
export const handler = async (event, context) => {
    console.log(JSON.stringify(event, null, 2));
    event.Records.forEach(record => {
        logDynamoDBRecord(record);
```

```
    });
}
const logDynamoDBRecord = (record) => {
    console.log(record.eventID);
    console.log(record.eventName);
    console.log(`DynamoDB Record: ${JSON.stringify(record.dynamodb)}`);
};
```

PHP

SDK untuk PHP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi peristiwa DynamoDB dengan Lambda menggunakan PHP.

```
<?php

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\DynamoDb\DynamoDbEvent;
use Bref\Event\DynamoDb\DynamoDbHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler extends DynamoDbHandler
{
    private StderrLogger $logger;

    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
```

```
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handleDynamoDb(DynamoDbEvent $event, Context $context): void
    {
        $this->logger->info("Processing DynamoDb table items");
        $records = $event->getRecords();

        foreach ($records as $record) {
            $eventName = $record->getEventName();
            $keys = $record->getKeys();
            $old = $record->getOldImage();
            $new = $record->getNewImage();

            $this->logger->info("Event Name:".$eventName."\n");
            $this->logger->info("Keys:". json_encode($keys)."\n");
            $this->logger->info("Old Image:". json_encode($old)."\n");
            $this->logger->info("New Image:". json_encode($new));

            // TODO: Do interesting work based on the new data

            // Any exception thrown will be logged and the invocation will be
 marked as failed
        }

        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords items");
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi peristiwa DynamoDB dengan Lambda menggunakan Python.

```
import json

def lambda_handler(event, context):
    print(json.dumps(event, indent=2))

    for record in event['Records']:
        log_dynamodb_record(record)

def log_dynamodb_record(record):
    print(record['eventID'])
    print(record['eventName'])
    print(f"DynamoDB Record: {json.dumps(record['dynamodb'])}")
```

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi peristiwa DynamoDB dengan Lambda menggunakan Ruby.

```
def lambda_handler(event:, context:)
    return 'received empty event' if event['Records'].empty?

    event['Records'].each do |record|
      log_dynamodb_record(record)
    end

    "Records processed: #{event['Records'].length}"
  end

  def log_dynamodb_record(record)
```

```
    puts record['eventID']
    puts record['eventName']
    puts "DynamoDB Record: #{JSON.generate(record['dynamodb'])}"
  end
```

Rust

SDK for Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi peristiwa DynamoDB dengan Lambda menggunakan Rust.

```
use lambda_runtime::{service_fn, tracing, Error, LambdaEvent};
use aws_lambda_events::{
    event::dynamodb::{Event, EventRecord},
   };


// Built with the following dependencies:
//lambda_runtime = "0.11.1"
//serde_json = "1.0"
//tokio = { version = "1", features = ["macros"] }
//tracing = { version = "0.1", features = ["log"] }
//tracing-subscriber = { version = "0.3", default-features = false, features =
 ["fmt"] }
//aws_lambda_events = "0.15.0"

async fn function_handler(event: LambdaEvent<Event>) ->Result<(), Error> {

    let records = &event.payload.records;
    tracing::info!("event payload: {:?}",records);
    if records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(());
    }
```

```
    for record in records{
        log_dynamo_dbrecord(record);
    }

    tracing::info!("Dynamo db records processed");

    // Prepare the response
    Ok(())

}

fn log_dynamo_dbrecord(record: &EventRecord)-> Result<(), Error>{
    tracing::info!("EventId: {}", record.event_id);
    tracing::info!("EventName: {}", record.event_name);
    tracing::info!("DynamoDB Record: {:?}", record.change );
    Ok(())

}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
    .with_max_level(tracing::Level::INFO)
    .with_target(false)
    .without_time()
    .init();

    let func = service_fn(function_handler);
    lambda_runtime::run(func).await?;
    Ok(())

}
```

Untuk daftar lengkap AWS Panduan pengembang SDK dan contoh kode, lihat Menggunakan Lambda dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Memanggil fungsi Lambda dari pemicu Amazon DocumentDB

Contoh kode berikut menunjukkan cara mengimplementasikan fungsi Lambda yang menerima peristiwa yang dipicu dengan menerima catatan dari aliran perubahan DocumentDB. Fungsi ini mengambil muatan DocumentDB dan mencatat konten rekaman.

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara Amazon DocumentDB dengan Lambda menggunakan.NET.

```
using Amazon.Lambda.Core;
using System.Text.Json;
using System;
using System.Collections.Generic;
using System.Text.Json.Serialization;
//Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri

namespace LambdaDocDb;

public class Function
{

     /// <summary>
    /// Lambda function entry point to process Amazon DocumentDB events.
    /// </summary>
    /// <param name="event">The Amazon DocumentDB event.</param>
    /// <param name="context">The Lambda context object.</param>
    /// <returns>A string to indicate successful processing.</returns>
    public string FunctionHandler(Event evnt, ILambdaContext context)
    {
```

```
        foreach (var record in evnt.Events)
        {
            ProcessDocumentDBEvent(record, context);
        }

        return "OK";
    }

     private void ProcessDocumentDBEvent(DocumentDBEventRecord record,
ILambdaContext context)
    {

        var eventData = record.Event;
        var operationType = eventData.OperationType;
        var databaseName = eventData.Ns.Db;
        var collectionName = eventData.Ns.Coll;
        var fullDocument = JsonSerializer.Serialize(eventData.FullDocument, new
JsonSerializerOptions { WriteIndented = true });

        context.Logger.LogLine($"Operation type: {operationType}");
        context.Logger.LogLine($"Database: {databaseName}");
        context.Logger.LogLine($"Collection: {collectionName}");
        context.Logger.LogLine($"Full document:\n{fullDocument}");
    }



    public class Event
    {
        [JsonPropertyName("eventSourceArn")]
        public string EventSourceArn { get; set; }

        [JsonPropertyName("events")]
        public List<DocumentDBEventRecord> Events { get; set; }

        [JsonPropertyName("eventSource")]
        public string EventSource { get; set; }
    }

    public class DocumentDBEventRecord
    {
        [JsonPropertyName("event")]
        public EventData Event { get; set; }
    }
```

```
public class EventData
{
    [JsonPropertyName("_id")]
    public IdData Id { get; set; }

    [JsonPropertyName("clusterTime")]
    public ClusterTime ClusterTime { get; set; }

    [JsonPropertyName("documentKey")]
    public DocumentKey DocumentKey { get; set; }

    [JsonPropertyName("fullDocument")]
    public Dictionary<string, object> FullDocument { get; set; }

    [JsonPropertyName("ns")]
    public Namespace Ns { get; set; }

    [JsonPropertyName("operationType")]
    public string OperationType { get; set; }
}

public class IdData
{
    [JsonPropertyName("_data")]
    public string Data { get; set; }
}

public class ClusterTime
{
    [JsonPropertyName("$timestamp")]
    public Timestamp Timestamp { get; set; }
}

public class Timestamp
{
    [JsonPropertyName("t")]
    public long T { get; set; }

    [JsonPropertyName("i")]
    public int I { get; set; }
}

public class DocumentKey
```

```
    {
        [JsonPropertyName("_id")]
        public Id Id { get; set; }
    }

    public class Id
    {
        [JsonPropertyName("$oid")]
        public string Oid { get; set; }
    }

    public class Namespace
    {
        [JsonPropertyName("db")]
        public string Db { get; set; }

        [JsonPropertyName("coll")]
        public string Coll { get; set; }
    }
}
```

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara Amazon DocumentDB dengan Lambda menggunakan Go.

```
package main

import (
 "context"
 "encoding/json"
 "fmt"
```

```
 "github.com/aws/aws-lambda-go/lambda"
)

type Event struct {
 Events []Record `json:"events"`
}

type Record struct {
 Event struct {
  OperationType string `json:"operationType"`
  NS            struct {
   DB   string `json:"db"`
   Coll string `json:"coll"`
  } `json:"ns"`
  FullDocument interface{} `json:"fullDocument"`
 } `json:"event"`
}

func main() {
 lambda.Start(handler)
}

func handler(ctx context.Context, event Event) (string, error) {
 fmt.Println("Loading function")
 for _, record := range event.Events {
  logDocumentDBEvent(record)
 }

 return "OK", nil
}

func logDocumentDBEvent(record Record) {
 fmt.Printf("Operation type: %s\n", record.Event.OperationType)
 fmt.Printf("db: %s\n", record.Event.NS.DB)
 fmt.Printf("collection: %s\n", record.Event.NS.Coll)
 docBytes, _ := json.MarshalIndent(record.Event.FullDocument, "", "  ")
 fmt.Printf("Full document: %s\n", string(docBytes))
}
```

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara Amazon DocumentDB dengan Lambda menggunakan Java.

```
import java.util.List;
import java.util.Map;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;

public class Example implements RequestHandler<Map<String, Object>, String> {

    @SuppressWarnings("unchecked")
    @Override
    public String handleRequest(Map<String, Object> event, Context context) {
        List<Map<String, Object>> events = (List<Map<String, Object>>)
 event.get("events");
        for (Map<String, Object> record : events) {
            Map<String, Object> eventData = (Map<String, Object>)
 record.get("event");
            processEventData(eventData);
        }

        return "OK";
    }

    @SuppressWarnings("unchecked")
    private void processEventData(Map<String, Object> eventData) {
        String operationType = (String) eventData.get("operationType");
        System.out.println("operationType: %s".formatted(operationType));

        Map<String, Object> ns = (Map<String, Object>) eventData.get("ns");

        String db = (String) ns.get("db");
        System.out.println("db: %s".formatted(db));
```

```
        String coll = (String) ns.get("coll");
        System.out.println("coll: %s".formatted(coll));

        Map<String, Object> fullDocument = (Map<String, Object>)
 eventData.get("fullDocument");
        System.out.println("fullDocument: %s".formatted(fullDocument));
    }

}
```

JavaScript

SDK untuk JavaScript (V3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara Amazon DocumentDB dengan menggunakan Lambda JavaScript.

```
console.log('Loading function');
exports.handler = async (event, context) => {
    event.events.forEach(record => {
        logDocumentDBEvent(record);
    });
    return 'OK';
};

const logDocumentDBEvent = (record) => {
    console.log('Operation type: ' + record.event.operationType);
    console.log('db: ' + record.event.ns.db);
    console.log('collection: ' + record.event.ns.coll);
    console.log('Full document:', JSON.stringify(record.event.fullDocument, null,
 2));
};
```

Mengkonsumsi acara Amazon DocumentDB dengan menggunakan Lambda TypeScript

```
import { DocumentDBEventRecord, DocumentDBEventSubscriptionContext } from 'aws-
lambda';

console.log('Loading function');

export const handler = async (
  event: DocumentDBEventSubscriptionContext,
  context: any
): Promise<string> => {
  event.events.forEach((record: DocumentDBEventRecord) => {
    logDocumentDBEvent(record);
  });
  return 'OK';
};

const logDocumentDBEvent = (record: DocumentDBEventRecord): void => {
  console.log('Operation type: ' + record.event.operationType);
  console.log('db: ' + record.event.ns.db);
  console.log('collection: ' + record.event.ns.coll);
  console.log('Full document:', JSON.stringify(record.event.fullDocument, null,
 2));
};
```

PHP

SDK untuk PHP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara Amazon DocumentDB dengan Lambda menggunakan PHP.

```
<?php

require __DIR__.'/vendor/autoload.php';
```

```
use Bref\Context\Context;
use Bref\Event\Handler;

class DocumentDBEventHandler implements Handler
{
    public function handle($event, Context $context): string
    {

        $events = $event['events'] ?? [];
        foreach ($events as $record) {
            $this->logDocumentDBEvent($record['event']);
        }
        return 'OK';
    }

    private function logDocumentDBEvent($event): void
    {
        // Extract information from the event record

        $operationType = $event['operationType'] ?? 'Unknown';
        $db = $event['ns']['db'] ?? 'Unknown';
        $collection = $event['ns']['coll'] ?? 'Unknown';
        $fullDocument = $event['fullDocument'] ?? [];

        // Log the event details

        echo "Operation type: $operationType\n";
        echo "Database: $db\n";
        echo "Collection: $collection\n";
        echo "Full document: " . json_encode($fullDocument, JSON_PRETTY_PRINT) .
 "\n";
    }
}
return new DocumentDBEventHandler();
```

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara Amazon DocumentDB dengan Lambda menggunakan Python.

```
import json

def lambda_handler(event, context):
    for record in event.get('events', []):
        log_document_db_event(record)
    return 'OK'

def log_document_db_event(record):
    event_data = record.get('event', {})
    operation_type = event_data.get('operationType', 'Unknown')
    db = event_data.get('ns', {}).get('db', 'Unknown')
    collection = event_data.get('ns', {}).get('coll', 'Unknown')
    full_document = event_data.get('fullDocument', {})

    print(f"Operation type: {operation_type}")
    print(f"db: {db}")
    print(f"collection: {collection}")
    print("Full document:", json.dumps(full_document, indent=2))
```

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara Amazon DocumentDB dengan Lambda menggunakan Ruby.

```
require 'json'

def lambda_handler(event:, context:)
  event['events'].each do |record|
    log_document_db_event(record)
  end
  'OK'
end

def log_document_db_event(record)
  event_data = record['event'] || {}
  operation_type = event_data['operationType'] || 'Unknown'
  db = event_data.dig('ns', 'db') || 'Unknown'
  collection = event_data.dig('ns', 'coll') || 'Unknown'
  full_document = event_data['fullDocument'] || {}

  puts "Operation type: #{operation_type}"
  puts "db: #{db}"
  puts "collection: #{collection}"
  puts "Full document: #{JSON.pretty_generate(full_document)}"
end
```

Rust

SDK for Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara Amazon DocumentDB dengan Lambda menggunakan Rust.

```
use lambda_runtime::{service_fn, tracing, Error, LambdaEvent};
use aws_lambda_events::{
    event::documentdb::{DocumentDbEvent, DocumentDbInnerEvent},
   };
```

```
// Built with the following dependencies:
//lambda_runtime = "0.11.1"
//serde_json = "1.0"
//tokio = { version = "1", features = ["macros"] }
//tracing = { version = "0.1", features = ["log"] }
//tracing-subscriber = { version = "0.3", default-features = false, features =
 ["fmt"] }
//aws_lambda_events = "0.15.0"

async fn function_handler(event: LambdaEvent<DocumentDbEvent>) ->Result<(),
 Error> {

    tracing::info!("Event Source ARN: {:?}", event.payload.event_source_arn);
    tracing::info!("Event Source: {:?}", event.payload.event_source);

    let records = &event.payload.events;

    if records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(());
    }

    for record in records{
        log_document_db_event(record);
    }

    tracing::info!("Document db records processed");

    // Prepare the response
    Ok(())

}

fn log_document_db_event(record: &DocumentDbInnerEvent)-> Result<(), Error>{
    tracing::info!("Change Event: {:?}", record.event);

    Ok(())

}

#[tokio::main]
async fn main() -> Result<(), Error> {
```

```
    tracing_subscriber::fmt()
    .with_max_level(tracing::Level::INFO)
    .with_target(false)
    .without_time()
    .init();

    let func = service_fn(function_handler);
    lambda_runtime::run(func).await?;
    Ok(())

}
```

Untuk daftar lengkap AWS Panduan pengembang SDK dan contoh kode, lihat Menggunakan Lambda dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Memanggil fungsi Lambda dari pemicu Amazon MSK

Contoh kode berikut menunjukkan cara mengimplementasikan fungsi Lambda yang menerima peristiwa yang dipicu dengan menerima catatan dari cluster Amazon MSK. Fungsi ini mengambil muatan MSK dan mencatat konten rekaman.

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara Amazon MSK dengan Lambda menggunakan .NET.

```
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.KafkaEvents;
```

```
// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri

namespace MSKLambda;

public class Function
{


    /// <param name="input">The event for the Lambda function handler to
 process.</param>
    /// <param name="context">The ILambdaContext that provides methods for
 logging and describing the Lambda environment.</param>
    /// <returns></returns>
    public void FunctionHandler(KafkaEvent evnt, ILambdaContext context)
    {

        foreach (var record in evnt.Records)
        {
            Console.WriteLine("Key:" + record.Key);
            foreach (var eventRecord in record.Value)
            {
                var valueBytes = eventRecord.Value.ToArray();
                var valueText = Encoding.UTF8.GetString(valueBytes);

                Console.WriteLine("Message:" + valueText);
            }
        }
    }


}
```

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara Amazon MSK dengan Lambda menggunakan Go.

```
package main

import (
 "encoding/base64"
 "fmt"

 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

func handler(event events.KafkaEvent) {
 for key, records := range event.Records {
  fmt.Println("Key:", key)

  for _, record := range records {
   fmt.Println("Record:", record)

   decodedValue, _ := base64.StdEncoding.DecodeString(record.Value)
   message := string(decodedValue)
   fmt.Println("Message:", message)
  }
 }
}

func main() {
 lambda.Start(handler)
}
```

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara Amazon MSK dengan Lambda menggunakan Java.

```
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.KafkaEvent;
import com.amazonaws.services.lambda.runtime.events.KafkaEvent.KafkaEventRecord;

import java.util.Base64;
import java.util.Map;

public class Example implements RequestHandler<KafkaEvent, Void> {

    @Override
    public Void handleRequest(KafkaEvent event, Context context) {
        for (Map.Entry<String, java.util.List<KafkaEventRecord>> entry :
 event.getRecords().entrySet()) {
            String key = entry.getKey();
            System.out.println("Key: " + key);

            for (KafkaEventRecord record : entry.getValue()) {
                System.out.println("Record: " + record);

                byte[] value = Base64.getDecoder().decode(record.getValue());
                String message = new String(value);
                System.out.println("Message: " + message);
            }
        }

        return null;
    }
}
```

JavaScript

SDK untuk JavaScript (V3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara Amazon MSK dengan menggunakan Lambda JavaScript.

```
exports.handler = async (event) => {
    // Iterate through keys
    for (let key in event.records) {
      console.log('Key: ', key)
      // Iterate through records
      event.records[key].map((record) => {
        console.log('Record: ', record)
        // Decode base64
        const msg = Buffer.from(record.value, 'base64').toString()
        console.log('Message:', msg)
      })
    }
}
```

Mengkonsumsi acara Amazon MSK dengan menggunakan Lambda TypeScript.

```
import { MSKEvent, Context } from "aws-lambda";
import { Buffer } from "buffer";
import { Logger } from "@aws-lambda-powertools/logger";

const logger = new Logger({
  logLevel: "INFO",
  serviceName: "msk-handler-sample",
});

export const handler = async (
```

```
  event: MSKEvent,
  context: Context
): Promise<void> => {
  for (const [topic, topicRecords] of Object.entries(event.records)) {
    logger.info(`Processing key: ${topic}`);

    // Process each record in the partition
    for (const record of topicRecords) {
      try {
        // Decode the message value from base64
        const decodedMessage = Buffer.from(record.value, 'base64').toString();

        logger.info({
          message: decodedMessage
        });
      }
      catch (error) {
        logger.error('Error processing event', { error });
        throw error;
      }
    };
  }
}
```

PHP

SDK untuk PHP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara Amazon MSK dengan Lambda menggunakan PHP.

```
<?php
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

// using bref/bref and bref/logger for simplicity
```

```
use Bref\Context\Context;
use Bref\Event\Kafka\KafkaEvent;
use Bref\Event\Handler as StdHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler implements StdHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handle(mixed $event, Context $context): void
    {
        $kafkaEvent = new KafkaEvent($event);
        $this->logger->info("Processing records");
        $records = $kafkaEvent->getRecords();

        foreach ($records as $record) {
            try {
                $key = $record->getKey();
                $this->logger->info("Key: $key");

                $values = $record->getValue();
                $this->logger->info(json_encode($values));

                foreach ($values as $value) {
                    $this->logger->info("Value: $value");
                }

            } catch (Exception $e) {
                $this->logger->error($e->getMessage());
            }
        }
        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords records");
```

```
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara Amazon MSK dengan Lambda menggunakan Python.

```
import base64

def lambda_handler(event, context):
    # Iterate through keys
    for key in event['records']:
        print('Key:', key)
        # Iterate through records
        for record in event['records'][key]:
            print('Record:', record)
            # Decode base64
            msg = base64.b64decode(record['value']).decode('utf-8')
            print('Message:', msg)
```

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara Amazon MSK dengan Lambda menggunakan Ruby.

```
require 'base64'

def lambda_handler(event:, context:)
  # Iterate through keys
  event['records'].each do |key, records|
    puts "Key: #{key}"

    # Iterate through records
    records.each do |record|
      puts "Record: #{record}"

      # Decode base64
      msg = Base64.decode64(record['value'])
      puts "Message: #{msg}"
    end
  end
end
```

Rust

SDK for Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara Amazon MSK dengan Lambda menggunakan Rust.

```
use aws_lambda_events::event::kafka::KafkaEvent;
use lambda_runtime::{run, service_fn, tracing, Error, LambdaEvent};
use base64::prelude::*;
use serde_json::{Value};
use tracing::{info};

/// Pre-Requisites:
/// 1. Install Cargo Lambda - see https://www.cargo-lambda.info/guide/getting-
started.html
/// 2. Add packages tracing, tracing-subscriber, serde_json, base64
///
/// This is the main body for the function.
/// Write your code inside it.
/// There are some code example in the following URLs:
/// - https://github.com/awslabs/aws-lambda-rust-runtime/tree/main/examples
/// - https://github.com/aws-samples/serverless-rust-demo/

async fn function_handler(event: LambdaEvent<KafkaEvent>) -> Result<Value, Error>
 {

    let payload = event.payload.records;

    for (_name, records) in payload.iter() {

        for record in records {

         let record_text = record.value.as_ref().ok_or("Value is None")?;
         info!("Record: {}", &record_text);

         // perform Base64 decoding
         let record_bytes = BASE64_STANDARD.decode(record_text)?;
         let message = std::str::from_utf8(&record_bytes)?;

         info!("Message: {}", message);
        }

    }

    Ok(().into())
}

#[tokio::main]
```

```
async fn main() -> Result<(), Error> {

    // required to enable CloudWatch error logging by the runtime
    tracing::init_default_subscriber();
    info!("Setup CW subscriber!");

    run(service_fn(function_handler)).await
}
```

Untuk daftar lengkap AWS Panduan pengembang SDK dan contoh kode, lihat Menggunakan Lambda dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Menginvokasi fungsi Lambda dari pemicu Amazon S3

Contoh kode berikut menunjukkan cara mengimplementasikan fungsi Lambda yang menerima peristiwa yang dipicu dengan mengunggah objek ke bucket S3. Fungsi ini mengambil nama bucket S3 dan kunci objek dari parameter peristiwa dan memanggil Amazon S3 API untuk mengambil dan mencatat jenis konten objek.

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Menggunakan peristiwa S3 dengan Lambda menggunakan.NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Threading.Tasks;
using Amazon.Lambda.Core;
using Amazon.S3;
using System;
using Amazon.Lambda.S3Events;
using System.Web;
```

```
// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri

namespace S3Integration
{
    public class Function
    {
        private static AmazonS3Client _s3Client;
        public Function() : this(null)
        {
        }

        internal Function(AmazonS3Client s3Client)
        {
            _s3Client = s3Client ?? new AmazonS3Client();
        }

        public async Task<string> Handler(S3Event evt, ILambdaContext context)
        {
            try
            {
                if (evt.Records.Count <= 0)
                {
                    context.Logger.LogLine("Empty S3 Event received");
                    return string.Empty;
                }

                var bucket = evt.Records[0].S3.Bucket.Name;
                var key = HttpUtility.UrlDecode(evt.Records[0].S3.Object.Key);

                context.Logger.LogLine($"Request is for {bucket} and {key}");

                var objectResult = await _s3Client.GetObjectAsync(bucket, key);

                context.Logger.LogLine($"Returning {objectResult.Key}");

                return objectResult.Key;
            }
            catch (Exception e)
            {
```

```
                context.Logger.LogLine($"Error processing request -
 {e.Message}");

                return string.Empty;
            }
        }
    }
}
```

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Menggunakan peristiwa S3 dengan Lambda menggunakan Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "log"

 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/s3"
)

func handler(ctx context.Context, s3Event events.S3Event) error {
 sdkConfig, err := config.LoadDefaultConfig(ctx)
 if err != nil {
  log.Printf("failed to load default config: %s", err)
  return err
 }
 s3Client := s3.NewFromConfig(sdkConfig)
```

```
 for _, record := range s3Event.Records {
  bucket := record.S3.Bucket.Name
  key := record.S3.Object.URLDecodedKey
  headOutput, err := s3Client.HeadObject(ctx, &s3.HeadObjectInput{
   Bucket: &bucket,
   Key:    &key,
  })
  if err != nil {
   log.Printf("error getting head of object %s/%s: %s", bucket, key, err)
   return err
  }
  log.Printf("successfully retrieved %s/%s of type %s", bucket, key,
 *headOutput.ContentType)
 }

 return nil
}

func main() {
 lambda.Start(handler)
}
```

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Menggunakan peristiwa S3 dengan Lambda menggunakan Java.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package example;

import software.amazon.awssdk.services.s3.model.HeadObjectRequest;
import software.amazon.awssdk.services.s3.model.HeadObjectResponse;
```

```
import software.amazon.awssdk.services.s3.S3Client;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.S3Event;
import
 com.amazonaws.services.lambda.runtime.events.models.s3.S3EventNotification.S3EventNotifi

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class Handler implements RequestHandler<S3Event, String> {
    private static final Logger logger = LoggerFactory.getLogger(Handler.class);
    @Override
    public String handleRequest(S3Event s3event, Context context) {
        try {
          S3EventNotificationRecord record = s3event.getRecords().get(0);
          String srcBucket = record.getS3().getBucket().getName();
          String srcKey = record.getS3().getObject().getUrlDecodedKey();

          S3Client s3Client = S3Client.builder().build();
          HeadObjectResponse headObject = getHeadObject(s3Client, srcBucket,
 srcKey);

          logger.info("Successfully retrieved " + srcBucket + "/" + srcKey + " of
 type " + headObject.contentType());

          return "Ok";
        } catch (Exception e) {
          throw new RuntimeException(e);
        }
    }

    private HeadObjectResponse getHeadObject(S3Client s3Client, String bucket,
 String key) {
        HeadObjectRequest headObjectRequest = HeadObjectRequest.builder()
                .bucket(bucket)
                .key(key)
                .build();
        return s3Client.headObject(headObjectRequest);
    }
}
```

JavaScript

SDK untuk JavaScript (V3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara S3 dengan menggunakan Lambda JavaScript.

```
import { S3Client, HeadObjectCommand } from "@aws-sdk/client-s3";

const client = new S3Client();

export const handler = async (event, context) => {

    // Get the object from the event and show its content type
    const bucket = event.Records[0].s3.bucket.name;
    const key = decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/g,
 ' '));

    try {
        const { ContentType } = await client.send(new HeadObjectCommand({
            Bucket: bucket,
            Key: key,
        }));

        console.log('CONTENT TYPE:', ContentType);
        return ContentType;

    } catch (err) {
        console.log(err);
        const message = `Error getting object ${key} from bucket ${bucket}. Make
 sure they exist and your bucket is in the same region as this function.`;
        console.log(message);
        throw new Error(message);
    }
};
```

Mengkonsumsi acara S3 dengan menggunakan Lambda TypeScript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { S3Event } from 'aws-lambda';
import { S3Client, HeadObjectCommand } from '@aws-sdk/client-s3';

const s3 = new S3Client({ region: process.env.AWS_REGION });

export const handler = async (event: S3Event): Promise<string | undefined> => {
  // Get the object from the event and show its content type
  const bucket = event.Records[0].s3.bucket.name;
  const key = decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/g, '
 '));
  const params = {
    Bucket: bucket,
    Key: key,
  };
  try {
    const { ContentType } = await s3.send(new HeadObjectCommand(params));
    console.log('CONTENT TYPE:', ContentType);
    return ContentType;
  } catch (err) {
    console.log(err);
    const message = `Error getting object ${key} from bucket ${bucket}. Make sure
 they exist and your bucket is in the same region as this function.`;
    console.log(message);
    throw new Error(message);
  }
};
```

PHP

SDK untuk PHP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara S3 dengan Lambda menggunakan PHP.

```
<?php

use Bref\Context\Context;
use Bref\Event\S3\S3Event;
use Bref\Event\S3\S3Handler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';


class Handler extends S3Handler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    public function handleS3(S3Event $event, Context $context) : void
    {
        $this->logger->info("Processing S3 records");

        // Get the object from the event and show its content type
        $records = $event->getRecords();

        foreach ($records as $record)
        {
            $bucket = $record->getBucket()->getName();
            $key = urldecode($record->getObject()->getKey());

            try {
                $fileSize = urldecode($record->getObject()->getSize());
                echo "File Size: " . $fileSize . "\n";
                // TODO: Implement your custom processing logic here
            } catch (Exception $e) {
                echo $e->getMessage() . "\n";
                echo 'Error getting object ' . $key . ' from bucket ' .
 $bucket . '. Make sure they exist and your bucket is in the same region as this
 function.' . "\n";
                throw $e;
            }
        }
```

```
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Menggunakan peristiwa S3 dengan Lambda menggunakan Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
import json
import urllib.parse
import boto3

print('Loading function')

s3 = boto3.client('s3')


def lambda_handler(event, context):
    #print("Received event: " + json.dumps(event, indent=2))

    # Get the object from the event and show its content type
    bucket = event['Records'][0]['s3']['bucket']['name']
    key = urllib.parse.unquote_plus(event['Records'][0]['s3']['object']['key'],
 encoding='utf-8')
    try:
        response = s3.get_object(Bucket=bucket, Key=key)
        print("CONTENT TYPE: " + response['ContentType'])
        return response['ContentType']
    except Exception as e:
```

```
        print(e)
        print('Error getting object {} from bucket {}. Make sure they exist and
 your bucket is in the same region as this function.'.format(key, bucket))
        raise e
```

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara S3 dengan Lambda menggunakan Ruby.

```
require 'json'
require 'uri'
require 'aws-sdk'

puts 'Loading function'

def lambda_handler(event:, context:)
  s3 = Aws::S3::Client.new(region: 'region') # Your AWS region
  # puts "Received event: #{JSON.dump(event)}"

  # Get the object from the event and show its content type
  bucket = event['Records'][0]['s3']['bucket']['name']
  key = URI.decode_www_form_component(event['Records'][0]['s3']['object']['key'],
 Encoding::UTF_8)
  begin
    response = s3.get_object(bucket: bucket, key: key)
    puts "CONTENT TYPE: #{response.content_type}"
    return response.content_type
  rescue StandardError => e
    puts e.message
    puts "Error getting object #{key} from bucket #{bucket}. Make sure they exist
 and your bucket is in the same region as this function."
    raise e
  end
```

```
end
```

Rust

SDK for Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Menggunakan peristiwa S3 dengan Lambda menggunakan Rust.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::event::s3::S3Event;
use aws_sdk_s3::{Client};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};


/// Main function
#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        .with_target(false)
        .without_time()
        .init();

    // Initialize the AWS SDK for Rust
    let config = aws_config::load_from_env().await;
    let s3_client = Client::new(&config);

    let res = run(service_fn(|request: LambdaEvent<S3Event>| {
        function_handler(&s3_client, request)
    })).await;

    res
}
```

```
async fn function_handler(
    s3_client: &Client,
    evt: LambdaEvent<S3Event>
) -> Result<(), Error> {
    tracing::info!(records = ?evt.payload.records.len(), "Received request from
 SQS");

    if evt.payload.records.len() == 0 {
        tracing::info!("Empty S3 event received");
    }

    let bucket = evt.payload.records[0].s3.bucket.name.as_ref().expect("Bucket
 name to exist");
    let key = evt.payload.records[0].s3.object.key.as_ref().expect("Object key to
 exist");

    tracing::info!("Request is for {} and object {}", bucket, key);

    let s3_get_object_result = s3_client
        .get_object()
        .bucket(bucket)
        .key(key)
        .send()
        .await;

    match s3_get_object_result {
        Ok(_) => tracing::info!("S3 Get Object success, the s3GetObjectResult
 contains a 'body' property of type ByteStream"),
        Err(_) => tracing::info!("Failure with S3 Get Object request")
    }

    Ok(())
}
```

Untuk daftar lengkap AWS Panduan pengembang SDK dan contoh kode, lihat Menggunakan Lambda dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Memanggil fungsi Lambda dari pemicu Amazon SNS

Contoh kode berikut menunjukkan cara mengimplementasikan fungsi Lambda yang menerima peristiwa yang dipicu dengan menerima pesan dari topik SNS. Fungsi mengambil pesan dari parameter peristiwa dan mencatat konten setiap pesan.

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara SNS dengan Lambda menggunakan.NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using Amazon.Lambda.Core;
using Amazon.Lambda.SNSEvents;


// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri

namespace SnsIntegration;

public class Function
{
    public async Task FunctionHandler(SNSEvent evnt, ILambdaContext context)
    {
        foreach (var record in evnt.Records)
        {
            await ProcessRecordAsync(record, context);
        }
        context.Logger.LogInformation("done");
    }
```

```
    private async Task ProcessRecordAsync(SNSEvent.SNSRecord record,
 ILambdaContext context)
    {
        try
        {
            context.Logger.LogInformation($"Processed record
 {record.Sns.Message}");

            // TODO: Do interesting work based on the new message
            await Task.CompletedTask;
        }
        catch (Exception e)
        {
            //You can use Dead Letter Queue to handle failures. By configuring a
 Lambda DLQ.
            context.Logger.LogError($"An error occurred");
            throw;
        }
    }
}
```

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara SNS dengan Lambda menggunakan Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "fmt"

 "github.com/aws/aws-lambda-go/events"
```

```
 "github.com/aws/aws-lambda-go/lambda"
)

func handler(ctx context.Context, snsEvent events.SNSEvent) {
 for _, record := range snsEvent.Records {
  processMessage(record)
 }
 fmt.Println("done")
}

func processMessage(record events.SNSEventRecord) {
 message := record.SNS.Message
 fmt.Printf("Processed message: %s\n", message)
 // TODO: Process your record here
}

func main() {
 lambda.Start(handler)
}
```

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara SNS dengan Lambda menggunakan Java.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.LambdaLogger;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.SNSEvent;
import com.amazonaws.services.lambda.runtime.events.SNSEvent.SNSRecord;
```

```
import java.util.Iterator;
import java.util.List;

public class SNSEventHandler implements RequestHandler<SNSEvent, Boolean> {
    LambdaLogger logger;

    @Override
    public Boolean handleRequest(SNSEvent event, Context context) {
        logger = context.getLogger();
        List<SNSRecord> records = event.getRecords();
        if (!records.isEmpty()) {
            Iterator<SNSRecord> recordsIter = records.iterator();
            while (recordsIter.hasNext()) {
                processRecord(recordsIter.next());
            }
        }
        return Boolean.TRUE;
    }

    public void processRecord(SNSRecord record) {
        try {
            String message = record.getSNS().getMessage();
            logger.log("message: " + message);
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }

}
```

JavaScript

SDK untuk JavaScript (V3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara SNS dengan menggunakan Lambda JavaScript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const record of event.Records) {
    await processMessageAsync(record);
  }
  console.info("done");
};

async function processMessageAsync(record) {
  try {
    const message = JSON.stringify(record.Sns.Message);
    console.log(`Processed message ${message}`);
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

Mengkonsumsi acara SNS dengan menggunakan Lambda TypeScript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { SNSEvent, Context, SNSHandler, SNSEventRecord } from "aws-lambda";

export const functionHandler: SNSHandler = async (
  event: SNSEvent,
  context: Context
```

```
): Promise<void> => {
  for (const record of event.Records) {
    await processMessageAsync(record);
  }
  console.info("done");
};

async function processMessageAsync(record: SNSEventRecord): Promise<any> {
  try {
    const message: string = JSON.stringify(record.Sns.Message);
    console.log(`Processed message ${message}`);
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

PHP

SDK untuk PHP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara SNS dengan Lambda menggunakan PHP.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

/*
Since native PHP support for AWS Lambda is not available, we are utilizing Bref's
 PHP functions runtime for AWS Lambda.
For more information on Bref's PHP runtime for Lambda, refer to: https://bref.sh/
docs/runtimes/function

Another approach would be to create a custom runtime.
```

```
A practical example can be found here: https://aws.amazon.com/blogs/apn/aws-
lambda-custom-runtime-for-php-a-practical-example/
*/

// Additional composer packages may be required when using Bref or any other PHP
 functions runtime.
// require __DIR__ . '/vendor/autoload.php';

use Bref\Context\Context;
use Bref\Event\Sns\SnsEvent;
use Bref\Event\Sns\SnsHandler;

class Handler extends SnsHandler
{
    public function handleSns(SnsEvent $event, Context $context): void
    {
        foreach ($event->getRecords() as $record) {
            $message = $record->getMessage();

            // TODO: Implement your custom processing logic here
            // Any exception thrown will be logged and the invocation will be
 marked as failed

            echo "Processed Message: $message" . PHP_EOL;
        }
    }
}

return new Handler();
```

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara SNS dengan Lambda menggunakan Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def lambda_handler(event, context):
    for record in event['Records']:
        process_message(record)
    print("done")

def process_message(record):
    try:
        message = record['Sns']['Message']
        print(f"Processed message {message}")
        # TODO; Process your record here

    except Exception as e:
        print("An error occurred")
        raise e
```

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara SNS dengan Lambda menggunakan Ruby.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def lambda_handler(event:, context:)
  event['Records'].map { |record| process_message(record) }
end

def process_message(record)
  message = record['Sns']['Message']
  puts("Processing message: #{message}")
rescue StandardError => e
  puts("Error processing message: #{e}")
```

```
  raise
end
```

Rust

SDK for Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara SNS dengan Lambda menggunakan Rust.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::event::sns::SnsEvent;
use aws_lambda_events::sns::SnsRecord;
use lambda_runtime::{run, service_fn, Error, LambdaEvent};
use tracing::info;

// Built with the following dependencies:
//  aws_lambda_events = { version = "0.10.0", default-features = false, features
 = ["sns"] }
//  lambda_runtime = "0.8.1"
//  tokio = { version = "1", features = ["macros"] }
//  tracing = { version = "0.1", features = ["log"] }
//  tracing-subscriber = { version = "0.3", default-features = false, features =
 ["fmt"] }

async fn function_handler(event: LambdaEvent<SnsEvent>) -> Result<(), Error> {
    for event in event.payload.records {
        process_record(&event)?;
    }

    Ok(())
}

fn process_record(record: &SnsRecord) -> Result<(), Error> {
    info!("Processing SNS Message: {}", record.sns.message);
```

```
    // Implement your record handling code here.

    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        .with_target(false)
        .without_time()
        .init();

    run(service_fn(function_handler)).await
}
```

Untuk daftar lengkap AWS Panduan pengembang SDK dan contoh kode, lihat Menggunakan Lambda dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Memanggil fungsi Lambda dari pemicu Amazon SQS

Contoh kode berikut menunjukkan bagaimana menerapkan fungsi Lambda yang menerima peristiwa yang dipicu dengan menerima pesan dari antrian SQS. Fungsi mengambil pesan dari parameter peristiwa dan mencatat konten setiap pesan.

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara SQS dengan Lambda menggunakan.NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
```

```
using Amazon.Lambda.Core;
using Amazon.Lambda.SQSEvents;


// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri

namespace SqsIntegrationSampleCode
{
    public async Task FunctionHandler(SQSEvent evnt, ILambdaContext context)
    {
        foreach (var message in evnt.Records)
        {
            await ProcessMessageAsync(message, context);
        }

        context.Logger.LogInformation("done");
    }

    private async Task ProcessMessageAsync(SQSEvent.SQSMessage message,
 ILambdaContext context)
    {
        try
        {
            context.Logger.LogInformation($"Processed message {message.Body}");

            // TODO: Do interesting work based on the new message
            await Task.CompletedTask;
        }
        catch (Exception e)
        {
            //You can use Dead Letter Queue to handle failures. By configuring a
 Lambda DLQ.
            context.Logger.LogError($"An error occurred");
            throw;
        }

    }
}
```

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara SQS dengan Lambda menggunakan Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package integration_sqs_to_lambda

import (
 "fmt"
 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

func handler(event events.SQSEvent) error {
 for _, record := range event.Records {
  err := processMessage(record)
  if err != nil {
   return err
  }
 }
 fmt.Println("done")
 return nil
}

func processMessage(record events.SQSMessage) error {
 fmt.Printf("Processed message %s\n", record.Body)
 // TODO: Do interesting work based on the new message
 return nil
}

func main() {
 lambda.Start(handler)
}
```

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara SQS dengan Lambda menggunakan Java.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.SQSEvent;
import com.amazonaws.services.lambda.runtime.events.SQSEvent.SQSMessage;

public class Function implements RequestHandler<SQSEvent, Void> {
    @Override
    public Void handleRequest(SQSEvent sqsEvent, Context context) {
        for (SQSMessage msg : sqsEvent.getRecords()) {
            processMessage(msg, context);
        }
        context.getLogger().log("done");
        return null;
    }

    private void processMessage(SQSMessage msg, Context context) {
        try {
            context.getLogger().log("Processed message " + msg.getBody());

            // TODO: Do interesting work based on the new message

        } catch (Exception e) {
            context.getLogger().log("An error occurred");
            throw e;
        }
```

```
    }
}
```

JavaScript

SDK untuk JavaScript (V3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara SQS dengan menggunakan Lambda JavaScript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const message of event.Records) {
    await processMessageAsync(message);
  }
  console.info("done");
};

async function processMessageAsync(message) {
  try {
    console.log(`Processed message ${message.body}`);
    // TODO: Do interesting work based on the new message
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

Mengkonsumsi acara SQS dengan menggunakan Lambda TypeScript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { SQSEvent, Context, SQSHandler, SQSRecord } from "aws-lambda";
```

```
export const functionHandler: SQSHandler = async (
  event: SQSEvent,
  context: Context
): Promise<void> => {
  for (const message of event.Records) {
    await processMessageAsync(message);
  }
  console.info("done");
};

async function processMessageAsync(message: SQSRecord): Promise<any> {
  try {
    console.log(`Processed message ${message.body}`);
    // TODO: Do interesting work based on the new message
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

PHP

SDK untuk PHP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara SQS dengan Lambda menggunakan PHP.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
```

```
use Bref\Event\InvalidLambdaEvent;
use Bref\Event\Sqs\SqsEvent;
use Bref\Event\Sqs\SqsHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler extends SqsHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws InvalidLambdaEvent
     */
    public function handleSqs(SqsEvent $event, Context $context): void
    {
        foreach ($event->getRecords() as $record) {
            $body = $record->getBody();
            // TODO: Do interesting work based on the new message
        }
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara SQS dengan Lambda menggunakan Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def lambda_handler(event, context):
    for message in event['Records']:
        process_message(message)
    print("done")

def process_message(message):
    try:
        print(f"Processed message {message['body']}")
        # TODO: Do interesting work based on the new message
    except Exception as err:
        print("An error occurred")
        raise err
```

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara SQS dengan Lambda menggunakan Ruby.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def lambda_handler(event:, context:)
  event['Records'].each do |message|
    process_message(message)
  end
  puts "done"
end

def process_message(message)
  begin
    puts "Processed message #{message['body']}"
    # TODO: Do interesting work based on the new message
```

```
  rescue StandardError => err
    puts "An error occurred"
    raise err
  end
end
```

Rust

SDK for Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Mengkonsumsi acara SQS dengan Lambda menggunakan Rust.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::event::sqs::SqsEvent;
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

async fn function_handler(event: LambdaEvent<SqsEvent>) -> Result<(), Error> {
    event.payload.records.iter().for_each(|record| {
        // process the record
        tracing::info!("Message body: {}",
 record.body.as_deref().unwrap_or_default())
    });

    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        // disable printing the name of the module in every log line.
        .with_target(false)
        // disabling time is handy because CloudWatch will add the ingestion
 time.
        .without_time()
```

```
        .init();

    run(service_fn(function_handler)).await
}
```

Untuk daftar lengkap AWS Panduan pengembang SDK dan contoh kode, lihat Menggunakan Lambda dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Melaporkan kegagalan item batch untuk fungsi Lambda dengan pemicu Kinesis

Contoh kode berikut menunjukkan bagaimana menerapkan respons batch parsional untuk fungsi Lambda yang menerima peristiwa dari aliran Kinesis. Fungsi melaporkan kegagalan item batch dalam respons, memberi sinyal ke Lambda untuk mencoba lagi pesan tersebut nanti.

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch Kinesis dengan Lambda menggunakan.NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text;
using System.Text.Json.Serialization;
using Amazon.Lambda.Core;
using Amazon.Lambda.KinesisEvents;
using AWS.Lambda.Powertools.Logging;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri
```

```
namespace KinesisIntegration;

public class Function
{
    // Powertools Logger requires an environment variables against your function
    // POWERTOOLS_SERVICE_NAME
    [Logging(LogEvent = true)]
    public async Task<StreamsEventResponse> FunctionHandler(KinesisEvent evnt,
 ILambdaContext context)
    {
        if (evnt.Records.Count == 0)
        {
            Logger.LogInformation("Empty Kinesis Event received");
            return new StreamsEventResponse();
        }

        foreach (var record in evnt.Records)
        {
            try
            {
                Logger.LogInformation($"Processed Event with EventId:
 {record.EventId}");
                string data = await GetRecordDataAsync(record.Kinesis, context);
                Logger.LogInformation($"Data: {data}");
                // TODO: Do interesting work based on the new data
            }
            catch (Exception ex)
            {
                Logger.LogError($"An error occurred {ex.Message}");
                /* Since we are working with streams, we can return the failed
 item immediately.
                   Lambda will immediately begin to retry processing from this
 failed item onwards. */
                return new StreamsEventResponse
                {
                    BatchItemFailures = new
 List<StreamsEventResponse.BatchItemFailure>
                    {
                        new StreamsEventResponse.BatchItemFailure
 { ItemIdentifier = record.Kinesis.SequenceNumber }
                    }
                };
            }
```

```
        }
        Logger.LogInformation($"Successfully processed {evnt.Records.Count}
 records.");
        return new StreamsEventResponse();
    }

    private async Task<string> GetRecordDataAsync(KinesisEvent.Record record,
 ILambdaContext context)
    {
        byte[] bytes = record.Data.ToArray();
        string data = Encoding.UTF8.GetString(bytes);
        await Task.CompletedTask; //Placeholder for actual async work
        return data;
    }
}

public class StreamsEventResponse
{
    [JsonPropertyName("batchItemFailures")]
    public IList<BatchItemFailure> BatchItemFailures { get; set; }
    public class BatchItemFailure
    {
        [JsonPropertyName("itemIdentifier")]
        public string ItemIdentifier { get; set; }
    }
}
```

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch Kinesis dengan Lambda menggunakan Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main
```

```
import (
 "context"
 "fmt"
 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

func handler(ctx context.Context, kinesisEvent events.KinesisEvent)
 (map[string]interface{}, error) {
 batchItemFailures := []map[string]interface{}{}

 for _, record := range kinesisEvent.Records {
  curRecordSequenceNumber := ""

  // Process your record
  if /* Your record processing condition here */ {
   curRecordSequenceNumber = record.Kinesis.SequenceNumber
  }

  // Add a condition to check if the record processing failed
  if curRecordSequenceNumber != "" {
   batchItemFailures = append(batchItemFailures, map[string]interface{}
{"itemIdentifier": curRecordSequenceNumber})
  }
 }

 kinesisBatchResponse := map[string]interface{}{
  "batchItemFailures": batchItemFailures,
 }
 return kinesisBatchResponse, nil
}

func main() {
 lambda.Start(handler)
}
```

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch Kinesis dengan Lambda menggunakan Java.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.KinesisEvent;
import com.amazonaws.services.lambda.runtime.events.StreamsEventResponse;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

public class ProcessKinesisRecords implements RequestHandler<KinesisEvent,
 StreamsEventResponse> {

    @Override
    public StreamsEventResponse handleRequest(KinesisEvent input, Context
 context) {

        List<StreamsEventResponse.BatchItemFailure> batchItemFailures = new
 ArrayList<>();
        String curRecordSequenceNumber = "";

        for (KinesisEvent.KinesisEventRecord kinesisEventRecord :
 input.getRecords()) {
            try {
                //Process your record
                KinesisEvent.Record kinesisRecord =
 kinesisEventRecord.getKinesis();
                curRecordSequenceNumber = kinesisRecord.getSequenceNumber();

            } catch (Exception e) {
```

```
                /* Since we are working with streams, we can return the failed
 item immediately.
                   Lambda will immediately begin to retry processing from this
 failed item onwards. */
                batchItemFailures.add(new
 StreamsEventResponse.BatchItemFailure(curRecordSequenceNumber));
                return new StreamsEventResponse(batchItemFailures);
            }
        }

       return new StreamsEventResponse(batchItemFailures);
    }
}
```

JavaScript

SDK untuk JavaScript (V3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch Kinesis dengan Lambda menggunakan Javascript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const record of event.Records) {
    try {
      console.log(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      console.log(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      console.error(`An error occurred ${err}`);
      /* Since we are working with streams, we can return the failed item
 immediately.
           Lambda will immediately begin to retry processing from this failed
 item onwards. */
```

```
      return {
        batchItemFailures: [{ itemIdentifier: record.kinesis.sequenceNumber }],
      };
    }
  }
  console.log(`Successfully processed ${event.Records.length} records.`);
  return { batchItemFailures: [] };
};

async function getRecordDataAsync(payload) {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

Melaporkan kegagalan item batch Kinesis dengan menggunakan Lambda TypeScript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  KinesisStreamEvent,
  Context,
  KinesisStreamHandler,
  KinesisStreamRecordPayload,
  KinesisStreamBatchResponse,
} from "aws-lambda";
import { Buffer } from "buffer";
import { Logger } from "@aws-lambda-powertools/logger";

const logger = new Logger({
  logLevel: "INFO",
  serviceName: "kinesis-stream-handler-sample",
});

export const functionHandler: KinesisStreamHandler = async (
  event: KinesisStreamEvent,
  context: Context
): Promise<KinesisStreamBatchResponse> => {
  for (const record of event.Records) {
    try {
      logger.info(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
```

```
      logger.info(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      logger.error(`An error occurred ${err}`);
      /* Since we are working with streams, we can return the failed item
 immediately.
           Lambda will immediately begin to retry processing from this failed
 item onwards. */
      return {
        batchItemFailures: [{ itemIdentifier: record.kinesis.sequenceNumber }],
      };
    }
  }
  logger.info(`Successfully processed ${event.Records.length} records.`);
  return { batchItemFailures: [] };
};

async function getRecordDataAsync(
  payload: KinesisStreamRecordPayload
): Promise<string> {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

PHP

SDK untuk PHP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch Kinesis dengan Lambda menggunakan PHP.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php
```

```
# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\Kinesis\KinesisEvent;
use Bref\Event\Handler as StdHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler implements StdHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handle(mixed $event, Context $context): array
    {
        $kinesisEvent = new KinesisEvent($event);
        $this->logger->info("Processing records");
        $records = $kinesisEvent->getRecords();

        $failedRecords = [];
        foreach ($records as $record) {
            try {
                $data = $record->getData();
                $this->logger->info(json_encode($data));
                // TODO: Do interesting work based on the new data
            } catch (Exception $e) {
                $this->logger->error($e->getMessage());
                // failed processing the record
                $failedRecords[] = $record->getSequenceNumber();
            }
        }
        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords records");

        // change format for the response
        $failures = array_map(
```

```
            fn(string $sequenceNumber) => ['itemIdentifier' => $sequenceNumber],
            $failedRecords
        );

        return [
            'batchItemFailures' => $failures
        ];
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch Kinesis dengan Lambda menggunakan Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def handler(event, context):
    records = event.get("Records")
    curRecordSequenceNumber = ""

    for record in records:
        try:
            # Process your record
            curRecordSequenceNumber = record["kinesis"]["sequenceNumber"]
        except Exception as e:
            # Return failed record's sequence number
            return {"batchItemFailures":[{"itemIdentifier":
 curRecordSequenceNumber}]}

    return {"batchItemFailures":[]}
```

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch Kinesis dengan Lambda menggunakan Ruby.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
require 'aws-sdk'

def lambda_handler(event:, context:)
  batch_item_failures = []

  event['Records'].each do |record|
    begin
      puts "Processed Kinesis Event - EventID: #{record['eventID']}"
      record_data = get_record_data_async(record['kinesis'])
      puts "Record Data: #{record_data}"
      # TODO: Do interesting work based on the new data
    rescue StandardError => err
      puts "An error occurred #{err}"
      # Since we are working with streams, we can return the failed item
 immediately.
      # Lambda will immediately begin to retry processing from this failed item
 onwards.
      return { batchItemFailures: [{ itemIdentifier: record['kinesis']
['sequenceNumber'] }] }
    end
  end

  puts "Successfully processed #{event['Records'].length} records."
  { batchItemFailures: batch_item_failures }
end
```

```
def get_record_data_async(payload)
  data = Base64.decode64(payload['data']).force_encoding('utf-8')
  # Placeholder for actual async work
  sleep(1)
  data
end
```

Rust

SDK for Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch Kinesis dengan Lambda menggunakan Rust.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::{
    event::kinesis::KinesisEvent,
    kinesis::KinesisEventRecord,
    streams::{KinesisBatchItemFailure, KinesisEventResponse},
};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

async fn function_handler(event: LambdaEvent<KinesisEvent>) ->
 Result<KinesisEventResponse, Error> {
    let mut response = KinesisEventResponse {
        batch_item_failures: vec![],
    };

    if event.payload.records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(response);
    }

    for record in &event.payload.records {
        tracing::info!(
            "EventId: {}",
```

```
            record.event_id.as_deref().unwrap_or_default()
        );

        let record_processing_result = process_record(record);

        if record_processing_result.is_err() {
            response.batch_item_failures.push(KinesisBatchItemFailure {
                item_identifier: record.kinesis.sequence_number.clone(),
            });
            /* Since we are working with streams, we can return the failed item
 immediately.
            Lambda will immediately begin to retry processing from this failed
 item onwards. */
            return Ok(response);
        }
    }

    tracing::info!(
        "Successfully processed {} records",
        event.payload.records.len()
    );

    Ok(response)
}

fn process_record(record: &KinesisEventRecord) -> Result<(), Error> {
    let record_data = std::str::from_utf8(record.kinesis.data.as_slice());

    if let Some(err) = record_data.err() {
        tracing::error!("Error: {}", err);
        return Err(Error::from(err));
    }

    let record_data = record_data.unwrap_or_default();

    // do something interesting with the data
    tracing::info!("Data: {}", record_data);

    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
```

```
        .with_max_level(tracing::Level::INFO)
        // disable printing the name of the module in every log line.
        .with_target(false)
        // disabling time is handy because CloudWatch will add the ingestion
 time.
        .without_time()
        .init();

    run(service_fn(function_handler)).await
}
```

Untuk daftar lengkap AWS Panduan pengembang SDK dan contoh kode, lihat Menggunakan Lambda dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Melaporkan kegagalan item batch untuk fungsi Lambda dengan pemicu DynamoDB

Contoh kode berikut menunjukkan bagaimana menerapkan respons batch parsional untuk fungsi Lambda yang menerima peristiwa dari aliran DynamoDB. Fungsi melaporkan kegagalan item batch dalam respons, memberi sinyal ke Lambda untuk mencoba lagi pesan tersebut nanti.

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch DynamoDB dengan Lambda menggunakan.NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text.Json;
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.DynamoDBEvents;
```

```
// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri

namespace AWSLambda_DDB;

public class Function
{
    public StreamsEventResponse FunctionHandler(DynamoDBEvent dynamoEvent,
 ILambdaContext context)

    {
        context.Logger.LogInformation($"Beginning to process
 {dynamoEvent.Records.Count} records...");
        List<StreamsEventResponse.BatchItemFailure> batchItemFailures = new
 List<StreamsEventResponse.BatchItemFailure>();
        StreamsEventResponse streamsEventResponse = new StreamsEventResponse();

        foreach (var record in dynamoEvent.Records)
        {
            try
            {
                var sequenceNumber = record.Dynamodb.SequenceNumber;
                context.Logger.LogInformation(sequenceNumber);
            }
            catch (Exception ex)
            {
                context.Logger.LogError(ex.Message);
                batchItemFailures.Add(new StreamsEventResponse.BatchItemFailure()
 { ItemIdentifier = record.Dynamodb.SequenceNumber });
            }
        }

        if (batchItemFailures.Count > 0)
        {
            streamsEventResponse.BatchItemFailures = batchItemFailures;
        }

        context.Logger.LogInformation("Stream processing complete.");
        return streamsEventResponse;
    }
```

```
}
```

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch DynamoDB dengan Lambda menggunakan Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

type BatchItemFailure struct {
 ItemIdentifier string `json:"ItemIdentifier"`
}

type BatchResult struct {
 BatchItemFailures []BatchItemFailure `json:"BatchItemFailures"`
}

func HandleRequest(ctx context.Context, event events.DynamoDBEvent)
 (*BatchResult, error) {
 var batchItemFailures []BatchItemFailure
 curRecordSequenceNumber := ""

 for _, record := range event.Records {
  // Process your record
  curRecordSequenceNumber = record.Change.SequenceNumber
 }
```

```
 if curRecordSequenceNumber != "" {
  batchItemFailures = append(batchItemFailures, BatchItemFailure{ItemIdentifier:
 curRecordSequenceNumber})
 }

 batchResult := BatchResult{
  BatchItemFailures: batchItemFailures,
 }

 return &batchResult, nil
}

func main() {
 lambda.Start(HandleRequest)
}
```

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch DynamoDB dengan Lambda menggunakan Java.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.DynamodbEvent;
import com.amazonaws.services.lambda.runtime.events.StreamsEventResponse;
import com.amazonaws.services.lambda.runtime.events.models.dynamodb.StreamRecord;

import java.util.ArrayList;
import java.util.List;

public class ProcessDynamodbRecords implements RequestHandler<DynamodbEvent,
 StreamsEventResponse> {
```

```
    @Override
    public StreamsEventResponse handleRequest(DynamodbEvent input, Context
 context) {

        List<StreamsEventResponse.BatchItemFailure> batchItemFailures = new
 ArrayList<>();
        String curRecordSequenceNumber = "";

        for (DynamodbEvent.DynamodbStreamRecord dynamodbStreamRecord :
 input.getRecords()) {
          try {
                //Process your record
                StreamRecord dynamodbRecord = dynamodbStreamRecord.getDynamodb();
                curRecordSequenceNumber = dynamodbRecord.getSequenceNumber();

            } catch (Exception e) {
                /* Since we are working with streams, we can return the failed
 item immediately.
                   Lambda will immediately begin to retry processing from this
 failed item onwards. */
                batchItemFailures.add(new
 StreamsEventResponse.BatchItemFailure(curRecordSequenceNumber));
                return new StreamsEventResponse(batchItemFailures);
            }
        }

       return new StreamsEventResponse();
    }
}
```

JavaScript

SDK untuk JavaScript (V3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch DynamoDB dengan menggunakan Lambda JavaScript.

```
export const handler = async (event) => {
  const records = event.Records;
  let curRecordSequenceNumber = "";

  for (const record of records) {
    try {
      // Process your record
      curRecordSequenceNumber = record.dynamodb.SequenceNumber;
    } catch (e) {
      // Return failed record's sequence number
      return { batchItemFailures: [{ itemIdentifier:
 curRecordSequenceNumber }] };
    }
  }

  return { batchItemFailures: [] };
};
```

Melaporkan kegagalan item batch DynamoDB dengan menggunakan Lambda TypeScript.

```
import {
  DynamoDBBatchResponse,
  DynamoDBBatchItemFailure,
  DynamoDBStreamEvent,
} from "aws-lambda";

export const handler = async (
  event: DynamoDBStreamEvent
): Promise<DynamoDBBatchResponse> => {
  const batchItemFailures: DynamoDBBatchItemFailure[] = [];
  let curRecordSequenceNumber;

  for (const record of event.Records) {
    curRecordSequenceNumber = record.dynamodb?.SequenceNumber;

    if (curRecordSequenceNumber) {
      batchItemFailures.push({
        itemIdentifier: curRecordSequenceNumber,
      });
    }
```

```
  }

  return { batchItemFailures: batchItemFailures };
};
```

PHP

SDK untuk PHP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch DynamoDB dengan Lambda menggunakan PHP.

```
<?php

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\DynamoDb\DynamoDbEvent;
use Bref\Event\Handler as StdHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler implements StdHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handle(mixed $event, Context $context): array
```

```
    {
        $dynamoDbEvent = new DynamoDbEvent($event);
        $this->logger->info("Processing records");

        $records = $dynamoDbEvent->getRecords();
        $failedRecords = [];
        foreach ($records as $record) {
            try {
                $data = $record->getData();
                $this->logger->info(json_encode($data));
                // TODO: Do interesting work based on the new data
            } catch (Exception $e) {
                $this->logger->error($e->getMessage());
                // failed processing the record
                $failedRecords[] = $record->getSequenceNumber();
            }
        }
        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords records");

        // change format for the response
        $failures = array_map(
            fn(string $sequenceNumber) => ['itemIdentifier' => $sequenceNumber],
            $failedRecords
        );

        return [
            'batchItemFailures' => $failures
        ];
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch DynamoDB dengan Lambda menggunakan Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def handler(event, context):
    records = event.get("Records")
    curRecordSequenceNumber = ""

    for record in records:
        try:
            # Process your record
            curRecordSequenceNumber = record["dynamodb"]["SequenceNumber"]
        except Exception as e:
            # Return failed record's sequence number
            return {"batchItemFailures":[{"itemIdentifier":
 curRecordSequenceNumber}]}

    return {"batchItemFailures":[]}
```

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch DynamoDB dengan Lambda menggunakan Ruby.

```
def lambda_handler(event:, context:)
    records = event["Records"]
    cur_record_sequence_number = ""

    records.each do |record|
      begin
        # Process your record
        cur_record_sequence_number = record["dynamodb"]["SequenceNumber"]
      rescue StandardError => e
        # Return failed record's sequence number
        return {"batchItemFailures" => [{"itemIdentifier" =>
 cur_record_sequence_number}]}
      end
    end

    {"batchItemFailures" => []}
  end
```

Rust

SDK for Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch DynamoDB dengan Lambda menggunakan Rust.

```
use aws_lambda_events::{
    event::dynamodb::{Event, EventRecord, StreamRecord},
    streams::{DynamoDbBatchItemFailure, DynamoDbEventResponse},
};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

/// Process the stream record
fn process_record(record: &EventRecord) -> Result<(), Error> {
    let stream_record: &StreamRecord = &record.change;
```

```
    // process your stream record here...
    tracing::info!("Data: {:?}", stream_record);

    Ok(())
}

/// Main Lambda handler here...
async fn function_handler(event: LambdaEvent<Event>) ->
 Result<DynamoDbEventResponse, Error> {
    let mut response = DynamoDbEventResponse {
        batch_item_failures: vec![],
    };

    let records = &event.payload.records;

    if records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(response);
    }

    for record in records {
        tracing::info!("EventId: {}", record.event_id);

        // Couldn't find a sequence number
        if record.change.sequence_number.is_none() {
            response.batch_item_failures.push(DynamoDbBatchItemFailure {
                item_identifier: Some("".to_string()),
            });
            return Ok(response);
        }

        // Process your record here...
        if process_record(record).is_err() {
            response.batch_item_failures.push(DynamoDbBatchItemFailure {
                item_identifier: record.change.sequence_number.clone(),
            });
            /* Since we are working with streams, we can return the failed item
 immediately.
            Lambda will immediately begin to retry processing from this failed
 item onwards. */
            return Ok(response);
        }
    }
```

```
    tracing::info!("Successfully processed {} record(s)", records.len());

    Ok(response)
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        // disable printing the name of the module in every log line.
        .with_target(false)
        // disabling time is handy because CloudWatch will add the ingestion
 time.
        .without_time()
        .init();

    run(service_fn(function_handler)).await
}
```

Untuk daftar lengkap AWS Panduan pengembang SDK dan contoh kode, lihat Menggunakan Lambda dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

Melaporkan kegagalan item batch untuk fungsi Lambda dengan pemicu Amazon SQS

Contoh kode berikut menunjukkan bagaimana menerapkan respons batch parsional untuk fungsi Lambda yang menerima peristiwa dari antrian SQS. Fungsi melaporkan kegagalan item batch dalam respons, memberi sinyal ke Lambda untuk mencoba lagi pesan tersebut nanti.

.NET

SDK untuk .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch SQS dengan Lambda menggunakan.NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using Amazon.Lambda.Core;
using Amazon.Lambda.SQSEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri
namespace sqsSample;

public class Function
{
    public async Task<SQSBatchResponse> FunctionHandler(SQSEvent evnt,
 ILambdaContext context)
    {
        List<SQSBatchResponse.BatchItemFailure> batchItemFailures = new
 List<SQSBatchResponse.BatchItemFailure>();
        foreach(var message in evnt.Records)
        {
            try
            {
                //process your message
                await ProcessMessageAsync(message, context);
            }
            catch (System.Exception)
            {
                //Add failed message identifier to the batchItemFailures list
                batchItemFailures.Add(new
 SQSBatchResponse.BatchItemFailure{ItemIdentifier=message.MessageId});
            }
        }
        return new SQSBatchResponse(batchItemFailures);
    }

    private async Task ProcessMessageAsync(SQSEvent.SQSMessage message,
 ILambdaContext context)
    {
        if (String.IsNullOrEmpty(message.Body))
        {
            throw new Exception("No Body in SQS Message.");
        }
```

```
        context.Logger.LogInformation($"Processed message {message.Body}");
        // TODO: Do interesting work based on the new message
        await Task.CompletedTask;
    }
}
```

Go

SDK untuk Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch SQS dengan Lambda menggunakan Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "fmt"
 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

func handler(ctx context.Context, sqsEvent events.SQSEvent)
 (map[string]interface{}, error) {
 batchItemFailures := []map[string]interface{}{}

 for _, message := range sqsEvent.Records {
  if len(message.Body) > 0 {
   // Your message processing condition here
   fmt.Printf("Successfully processed message: %s\n", message.Body)
  } else {
   // Message processing failed
   fmt.Printf("Failed to process message %s\n", message.MessageId)
   batchItemFailures = append(batchItemFailures, map[string]interface{}
{"itemIdentifier": message.MessageId})
```

```
  }
 }

 sqsBatchResponse := map[string]interface{}{
  "batchItemFailures": batchItemFailures,
 }
 return sqsBatchResponse, nil
}

func main() {
 lambda.Start(handler)
}
```

Java

SDK untuk Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch SQS dengan Lambda menggunakan Java.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.SQSEvent;
import com.amazonaws.services.lambda.runtime.events.SQSBatchResponse;

import java.util.ArrayList;
import java.util.List;

public class ProcessSQSMessageBatch implements RequestHandler<SQSEvent,
 SQSBatchResponse> {
    @Override
    public SQSBatchResponse handleRequest(SQSEvent sqsEvent, Context context) {
         List<SQSBatchResponse.BatchItemFailure> batchItemFailures = new
 ArrayList<SQSBatchResponse.BatchItemFailure>();
```

```
        for (SQSEvent.SQSMessage message : sqsEvent.getRecords()) {
            try {
                //process your message
            } catch (Exception e) {
                //Add failed message identifier to the batchItemFailures list
                batchItemFailures.add(new
 SQSBatchResponse.BatchItemFailure(message.getMessageId()));
            }
        }
        return new SQSBatchResponse(batchItemFailures);
    }
}
```

JavaScript

SDK untuk JavaScript (V3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch SQS dengan menggunakan Lambda JavaScript.

```
// Node.js 20.x Lambda runtime, AWS SDK for Javascript V3
export const handler = async (event, context) => {
    const batchItemFailures = [];
    for (const record of event.Records) {
        try {
            await processMessageAsync(record, context);
        } catch (error) {
            batchItemFailures.push({ itemIdentifier: record.messageId });
        }
    }
    return { batchItemFailures };
};

async function processMessageAsync(record, context) {
    if (record.body && record.body.includes("error")) {
        throw new Error("There is an error in the SQS Message.");
```

```
    }
    console.log(`Processed message: ${record.body}`);
}
```

Melaporkan kegagalan item batch SQS dengan menggunakan Lambda TypeScript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { SQSEvent, SQSBatchResponse, Context, SQSBatchItemFailure, SQSRecord }
 from 'aws-lambda';

export const handler = async (event: SQSEvent, context: Context):
 Promise<SQSBatchResponse> => {
    const batchItemFailures: SQSBatchItemFailure[] = [];

    for (const record of event.Records) {
        try {
            await processMessageAsync(record);
        } catch (error) {
            batchItemFailures.push({ itemIdentifier: record.messageId });
        }
    }

    return {batchItemFailures: batchItemFailures};
};

async function processMessageAsync(record: SQSRecord): Promise<void> {
    if (record.body && record.body.includes("error")) {
        throw new Error('There is an error in the SQS Message.');
    }
    console.log(`Processed message ${record.body}`);
}
```

PHP

SDK untuk PHP

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch SQS dengan Lambda menggunakan PHP.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

use Bref\Context\Context;
use Bref\Event\Sqs\SqsEvent;
use Bref\Event\Sqs\SqsHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler extends SqsHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handleSqs(SqsEvent $event, Context $context): void
    {
        $this->logger->info("Processing SQS records");
        $records = $event->getRecords();

        foreach ($records as $record) {
            try {
                // Assuming the SQS message is in JSON format
```

```
                $message = json_decode($record->getBody(), true);
                $this->logger->info(json_encode($message));
                // TODO: Implement your custom processing logic here
            } catch (Exception $e) {
                $this->logger->error($e->getMessage());
                // failed processing the record
                $this->markAsFailed($record);
            }
        }
        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords SQS records");
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

Python

SDK untuk Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch SQS dengan Lambda menggunakan Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0

def lambda_handler(event, context):
    if event:
        batch_item_failures = []
        sqs_batch_response = {}

        for record in event["Records"]:
            try:
                print(f"Processed message: {record['body']}")
            except Exception as e:
```

```
                batch_item_failures.append({"itemIdentifier":
 record['messageId']})

        sqs_batch_response["batchItemFailures"] = batch_item_failures
        return sqs_batch_response
```

Ruby

SDK untuk Ruby

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch SQS dengan Lambda menggunakan Ruby.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
require 'json'

def lambda_handler(event:, context:)
  if event
    batch_item_failures = []
    sqs_batch_response = {}

    event["Records"].each do |record|
      begin
        # process message
      rescue StandardError => e
        batch_item_failures << {"itemIdentifier" => record['messageId']}
      end
    end

    sqs_batch_response["batchItemFailures"] = batch_item_failures
    return sqs_batch_response
  end
end
```

Rust

SDK for Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di repositori contoh Nirserver.

Melaporkan kegagalan item batch SQS dengan Lambda menggunakan Rust.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::{
    event::sqs::{SqsBatchResponse, SqsEvent},
    sqs::{BatchItemFailure, SqsMessage},
};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

async fn process_record(_: &SqsMessage) -> Result<(), Error> {
    Err(Error::from("Error processing message"))
}

async fn function_handler(event: LambdaEvent<SqsEvent>) ->
 Result<SqsBatchResponse, Error> {
    let mut batch_item_failures = Vec::new();
    for record in event.payload.records {
        match process_record(&record).await {
            Ok(_) => (),
            Err(_) => batch_item_failures.push(BatchItemFailure {
                item_identifier: record.message_id.unwrap(),
            }),
        }
    }

    Ok(SqsBatchResponse {
        batch_item_failures,
    })
}

#[tokio::main]
async fn main() -> Result<(), Error> {
```

```
    run(service_fn(function_handler)).await
}
```

Untuk daftar lengkap AWS Panduan pengembang SDK dan contoh kode, lihat Menggunakan Lambda dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

AWS kontribusi komunitas untuk Lambda

AWS kontribusi masyarakat adalah contoh yang dibuat dan dikelola oleh beberapa tim di seluruh AWS. Untuk memberikan umpan balik, gunakan mekanisme yang disediakan di repositori tertaut.

Contoh

- Membangun dan menguji aplikasi tanpa server

Membangun dan menguji aplikasi tanpa server

Contoh kode berikut menunjukkan cara membangun dan menguji aplikasi tanpa server menggunakan API Gateway dengan Lambda dan DynamoDB

.NET

SDK untuk .NET

Menunjukkan cara membangun dan menguji aplikasi tanpa server yang terdiri dari API Gateway dengan Lambda dan DynamoDB menggunakan SDK.NET.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkap di GitHub.

Layanan yang digunakan dalam contoh ini

- API Gateway
- DynamoDB
- Lambda

Go

SDK untuk Go V2

Menunjukkan cara membangun dan menguji aplikasi tanpa server yang terdiri dari API Gateway dengan Lambda dan DynamoDB menggunakan Go SDK.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkap di GitHub.

Layanan yang digunakan dalam contoh ini

- API Gateway
- DynamoDB
- Lambda

Java

SDK untuk Java 2.x

Menunjukkan cara membangun dan menguji aplikasi tanpa server yang terdiri dari API Gateway dengan Lambda dan DynamoDB menggunakan Java SDK.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkap di GitHub.

Layanan yang digunakan dalam contoh ini

- API Gateway
- DynamoDB
- Lambda

Rust

SDK for Rust

Menunjukkan cara membangun dan menguji aplikasi tanpa server yang terdiri dari API Gateway dengan Lambda dan DynamoDB menggunakan Rust SDK.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkap di GitHub.

Layanan yang digunakan dalam contoh ini

- API Gateway
- DynamoDB
- Lambda

Untuk daftar lengkap AWS Panduan pengembang SDK dan contoh kode, lihat Menggunakan Lambda dengan AWS SDK. Topik ini juga mencakup informasi tentang memulai dan detail tentang versi SDK sebelumnya.

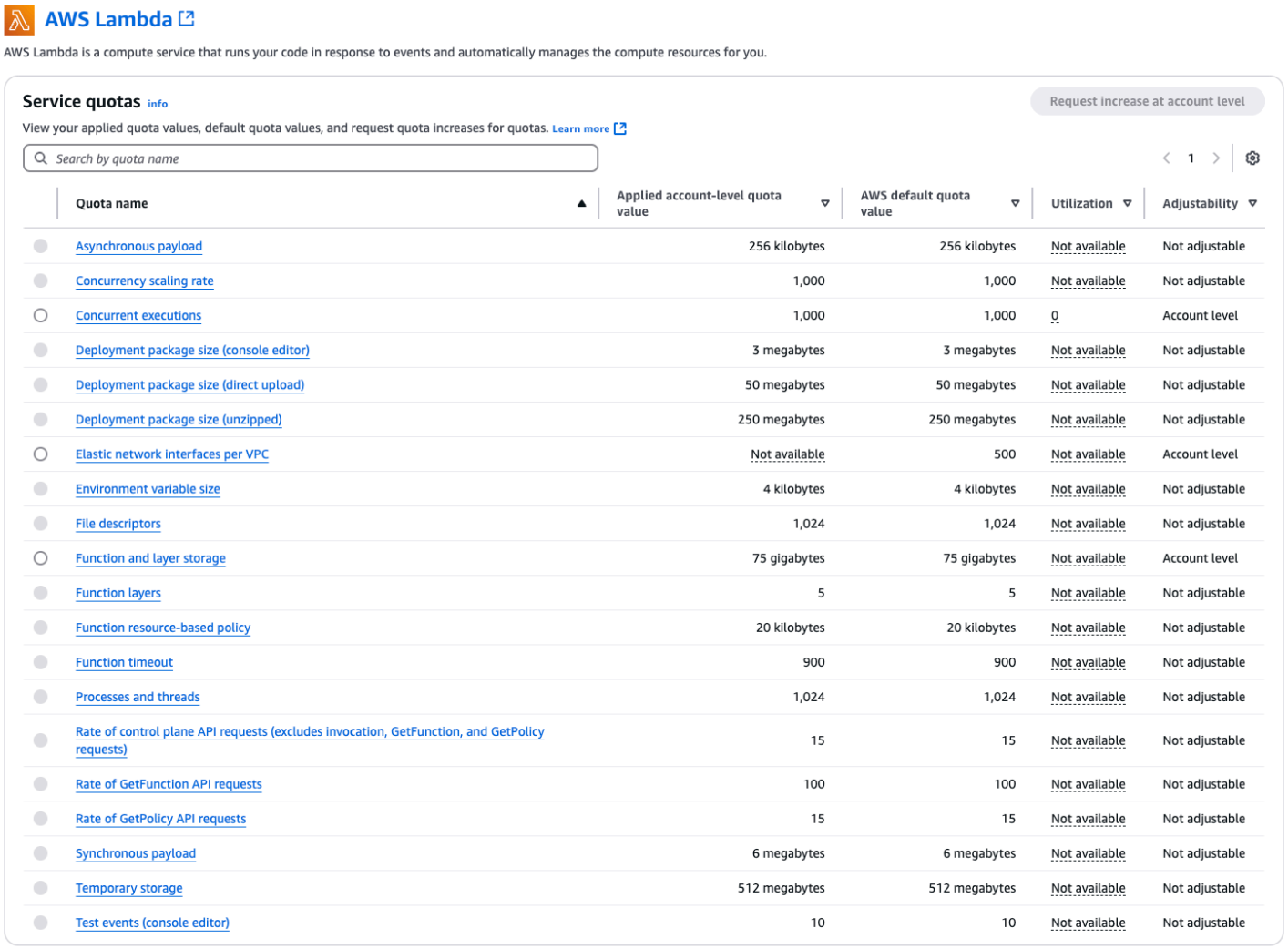
Kuota Lambda

> **Important**
>
> Baru Akun AWS telah mengurangi konkurensi dan kuota memori untuk Lambda Functions dan Lambda MicroVM, dan mengurangi vCPU dan kuota laju permintaan untuk Lambda Web Functions. AWS meningkatkan kuota ini secara otomatis berdasarkan penggunaan Anda.

AWS Lambda dirancang untuk menskalakan dengan cepat untuk memenuhi permintaan, memungkinkan fungsi Anda ditingkatkan untuk melayani lalu lintas di aplikasi Anda. Lambda dirancang untuk tugas komputasi berumur pendek yang tidak mempertahankan atau bergantung pada status di antara pemanggilan. Kode dapat berjalan hingga 15 menit dalam satu pemanggilan (hingga 90 menit untuk fungsi Instans Terkelola Lambda yang dipanggil secara asinkron atau melalui pemetaan sumber peristiwa apa pun kecuali Amazon MQ dan Amazon DocumentDB) dan satu fungsi dapat menggunakan memori hingga 10.240 MB.

Penting untuk memahami pagar pembatas yang dipasang untuk melindungi akun Anda dan beban kerja pelanggan lain. Kuota layanan ada di semua AWS layanan dan terdiri dari batas keras, yang tidak dapat Anda ubah, dan batas lunak, yang dapat Anda minta kenaikannya. Secara default, semua akun baru diberi profil kuota yang memungkinkan eksplorasi AWS layanan.

Untuk melihat kuota yang berlaku untuk akun Anda, navigasikan ke Dasbor Kuota Layanan. Di sini, Anda dapat melihat kuota layanan Anda, meminta kenaikan kuota, dan melihat pemanfaatan saat ini. Dari sini, Anda dapat menelusuri ke yang spesifik AWS layanan, seperti Lambda:



AWS Lambda

AWS Lambda is a compute service that runs your code in response to events and automatically manages the compute resources for you.

Service quotas info

Request increase at account level

View your applied quota values, default quota values, and request quota increases for quotas. Learn more

Search by quota name

1

Quota name	Applied account-level quota value	AWS default quota value	Utilization	Adjustability
Asynchronous payload	256 kilobytes	256 kilobytes	Not available	Not adjustable
Concurrency scaling rate	1,000	1,000	Not available	Not adjustable
Concurrent executions	1,000	1,000	0	Account level
Deployment package size (console editor)	3 megabytes	3 megabytes	Not available	Not adjustable
Deployment package size (direct upload)	50 megabytes	50 megabytes	Not available	Not adjustable
Deployment package size (unzipped)	250 megabytes	250 megabytes	Not available	Not adjustable
Elastic network interfaces per VPC	Not available	500	Not available	Account level
Environment variable size	4 kilobytes	4 kilobytes	Not available	Not adjustable
File descriptors	1,024	1,024	Not available	Not adjustable
Function and layer storage	75 gigabytes	75 gigabytes	Not available	Account level
Function layers	5	5	Not available	Not adjustable
Function resource-based policy	20 kilobytes	20 kilobytes	Not available	Not adjustable
Function timeout	900	900	Not available	Not adjustable
Processes and threads	1,024	1,024	Not available	Not adjustable
Rate of control plane API requests (excludes invocation, GetFunction, and GetPolicy requests)	15	15	Not available	Not adjustable
Rate of GetFunction API requests	100	100	Not available	Not adjustable
Rate of GetPolicy API requests	15	15	Not available	Not adjustable
Synchronous payload	6 megabytes	6 megabytes	Not available	Not adjustable
Temporary storage	512 megabytes	512 megabytes	Not available	Not adjustable
Test events (console editor)	10	10	Not available	Not adjustable

Bagian berikut mencantumkan kuota dan batas default di Lambda berdasarkan kategori.

Topik

- Komputasi dan penyimpanan
- Konfigurasi fungsi, penerapan, dan eksekusi
- Permintaan API Lambda
- Fungsi tahan lama
- MicroVM Lambda
- Fungsi Web Lambda
- Layanan lainnya

Komputasi dan penyimpanan

Lambda menetapkan kuota untuk jumlah sumber daya komputasi dan penyimpanan yang dapat Anda gunakan untuk menjalankan dan menyimpan fungsi. Kuota untuk eksekusi dan penyimpanan bersamaan berlaku per Wilayah AWS. Kuota antarmuka jaringan elastis (ENI) berlaku per cloud pribadi virtual (VPC), terlepas dari Wilayah. Kuota berikut dapat ditingkatkan dari nilai defaultnya. Untuk informasi selengkapnya, lihat Meminta peningkatan kuota dalam Panduan Pengguna Service Quotas.

Sumber Daya	Kuota default	Dapat ditingkatkan hingga
Eksekusi bersamaan	1.000	Puluhan ribu
Penyimpanan untuk fungsi yang diunggah (arsip file.zip) dan lapisan menggunakan Lambda-managed penyimpanan. Setiap versi fungsi dan versi lapisan mengonsumsi penyimpanan. Untuk menghindari batas penyimpanan, Anda dapat mengonfigurasi fungsi dan lapisan untuk digunakan penyimpanan kode S3 yang dikelola sendiri sebagai gantinya. Untuk praktik terbaik dalam mengelola penyimpanan kode Anda, lihat Memantau penyimpanan kode Lambda di Tanah Tanpa Server.	300 GB (dibuka ritsleting)	Tidak dapat ditingkatkan. Gunakan penyimpanan kode S3 yang dikelola sendiri untuk penyimpanan di luar batas ini.
Penyimpanan untuk fungsi yang ditentukan sebagai gambar kontainer. Gambar-gambar ini disimpan di Amazon ECR.	Lihat Kuota layanan Amazon ECR.	
Antarmuka jaringan elastis per cloud pribadi virtual (VPC)	3000	Ribuan

Sumber Daya	Kuota default	Dapat ditingkatkan hingga
Note Kuota ini dibagikan dengan layanan lainnya, seperti Amazon Elastic File System (Amazon EFS). Lihat Kuota Amazon VPC		

Untuk detail tentang konkurensi dan bagaimana Lambda menskalakan konkurensi fungsi Anda sehubungan dengan lalu lintas, lihat Memahami penskalaan Fungsi Acara Lambda.

Konfigurasi fungsi, penerapan, dan eksekusi

Kuota berikut berlaku untuk konfigurasi fungsi, penerapan, dan eksekusi. Kecuali seperti yang disebutkan, mereka tidak dapat diubah.

Note

Dokumentasi Lambda, pesan log, dan konsol menggunakan singkatan MB (bukan MiB) untuk merujuk ke 1.024 KB.

Sumber daya	Kuota
Fungsi alokasi memori	128 MB hingga 10.240 MB, dengan peningkatan 1 MB. Catatan: Lambda mengalokasikan daya CPU sebanding dengan jumlah memori yang dikonfigu rasi. Anda dapat menambah atau mengurangi memori dan daya CPU yang diberikan untuk fungsi Anda

Sumber daya	Kuota
	menggunakan pengaturan Memori (MB). Pada 1.769 MB, fungsi setara dengan satu vCPU.
Fungsi waktu habis	900 detik (15 menit). Untuk fungsi yang menggunakan AWS Lambda Instans Terkelola, pemanggil an asinkron, dan pemanggil an pemetaan sumber peristiwa (kecuali Amazon MQ dan Amazon DocumentDB) mendukung nilai maksimum yang diizinkan yaitu 5.400 detik (90 menit).
Fungsi variabel lingkungan	4 KB, untuk semua variabel lingkungan yang terkait dengan fungsi, secara agregat
Fungsi kebijakan berbasis sumber daya	20 KB
Fungsi lapisan	5 lapisan
Fungsi batas penskalaan konkurensi	Untuk setiap fungsi, 1.000 lingkungan eksekusi setiap 10 detik
Muatan invokasi (permintaan dan respons)	6 MB masing-masing untuk permintaan dan respons (sinkron) 200 MB untuk masing-masing respon yang dialirkan (sinkron) 1 MB (asinkron) 1 MB untuk ukuran gabungan total nilai baris permintaan dan header

Sumber daya	Kuota
Bandwidth untuk tanggapan yang dialirkan	Tidak dibatasi untuk 6 MB pertama dari respons fungsi Anda Untuk tanggapan yang lebih besar dari 6 MB, 2MBps untuk sisa respons
Bandwidth jaringan per lingkungan eksekusi	625 Mbps. Untuk fungsi yang tidak melekat pada VPC, Anda dapat meminta peningkatan melalui Konsol Kuota Layanan. Setelah disetujui, bandwidth jaringan diskalakan secara proporsional dengan memori, mulai dari memori 2.048 MB dan mencapai maksimum hingga 3.000 Mbps pada 10.240 MB.
Ukuran paket deployment (arsip file .zip)	50 MB (di-zip, saat diunggah melalui API Lambda atau SDK). Unggah file yang lebih besar dengan Amazon S3. 50 MB (saat diunggah melalui konsol Lambda) 250 MB Ukuran maksimum konten paket penerapan, termasuk lapisan dan runtime khusus. (dibuka ritsletin g)
Ukuran pengaturan gambar kontainer	16 KB
Ukuran paket kode gambar kontainer	10 GB (ukuran gambar tidak terkompresi maksimum, termasuk semua lapisan)
Peristiwa pengujian (editor konsol)	10

Sumber daya	Kuota
`/tmp` penyimpanan direktori	Antara 512 MB dan 10.240 MB, dengan penambahan 1 MB
Deskriptor file	1.024 **Note** Instans Terkelola Lambda menggunakan batas deskriptor file yang lebih tinggi yaitu 4.096. Untuk informasi selengkapnya, lihat Memahami lingkungan eksekusi Instans Terkelola Lambda.
Eksekusi processes/threads	1.024 **Note** Instans Terkelola Lambda menggunakan proses default dan batas utas dari Bottlerocket. Untuk informasi selengkapnya, lihat Memahami lingkungan eksekusi Instans Terkelola Lambda.

Permintaan API Lambda

Kuota berikut berkaitan dengan permintaan API Lambda.

Sumber Daya	Kuota
Permintaan pemanggilan per fungsi per Wilayah (sinkron)	Setiap instance lingkungan eksekusi Anda dapat melayani hingga 10 permintaan per detik. Dengan kata lain, batas pemanggilan total adalah 10 kali batas konkurensi Anda. Lihat Memahami penskalaan Fungsi Acara Lambda.
Permintaan pemanggilan per fungsi per Wilayah (asinkron)	Setiap instance lingkungan eksekusi Anda dapat melayani jumlah permintaan yang tidak terbatas. Dengan kata lain, batas pemanggil an total hanya didasarkan pada konkurensi yang tersedia untuk fungsi Anda. Lihat Memahami penskalaan Fungsi Acara Lambda.
Permintaan invokasi per versi fungsi atau alias (perminta an per detik)	10 x dialokasikan konkurensi tersedia **Note** Kuota ini hanya berlaku untuk fungsi yang menggunakan konkurensi terprovisi.
GetFunction Permintaan API	100 permintaan per detik. Tidak bisa ditingkatkan.
GetPolicy Permintaan API	15 permintaan per detik. Tidak bisa ditingkatkan.
Sisa permintaan API bidang kontrol (tidak termasuk pemanggilan, GetFunction, dan GetPolicy permintaan)	15 permintaan per detik di semua API (bukan 15 permintaan per detik per API). Tidak bisa ditingkatkan.

Fungsi tahan lama

Kuota berikut berlaku untuk fungsi tahan lama. Kuota berlaku per Wilayah AWS. Default yang lebih tinggi berlaku di AS Timur (Virginia Utara), AS Barat (Oregon), dan Eropa (Irlandia), seperti yang tercantum dalam tabel. Kecuali seperti yang disebutkan, kuota ini dapat ditingkatkan. Untuk informasi selengkapnya, lihat Meminta peningkatan kuota dalam Panduan Pengguna Service Quotas.

Sumber Daya	Kuota bawaan
Eksekusi tahan lama yang berjalan maksimal	5.000.000 per Wilayah. 10.000.000 di AS Timur (Virginia Utara), AS Barat (Oregon), dan Eropa (Irlandia). Ini adalah batas lunak yang dapat ditingkatkan.
Jumlah maksimum operasi tahan lama per eksekusi yang tahan lama	30,000 Tidak bisa ditingkatkan. Untuk informasi lebih lanjut, lihat Operasi tahan lama yang tersedia.
Penyimpanan eksekusi tahan lama ditulis dalam megabyte	100 MB Tidak bisa ditingkatkan. Ukuran payload kumulatif dipertahankan oleh fungsi tahan lama per eksekusi. Untuk informasi lebih lanjut, lihat data yang bertahan per operasi yang tahan lama.
Ukuran muatan maksimum per operasi yang tahan lama	1 MB Tidak bisa ditingkatkan. Berlaku untuk payload checkpoint dan callback.

Sumber Daya	Kuota bawaan
Tingkat pemanggilan fungsi untuk memulai eksekusi yang tahan lama	3.000 permintaan per detik. 6.000 permintaan per detik di AS Timur (Virginia Utara), AS Barat (Oregon), dan Eropa (Irlandia).
CheckpointDurableExecution Permintaan API	10.000 permintaan per detik. 20.000 permintaan per detik di AS Timur (Virginia Utara), AS Barat (Oregon), dan Eropa (Irlandia).
GetDurableExecution Permintaan API	30 permintaan per detik.
GetDurableExecutionHistory Permintaan API	15 permintaan per detik.
GetDurableExecutionState Permintaan API	10.000 permintaan per detik. 20.000 permintaan per detik di AS Timur (Virginia Utara), AS Barat (Oregon), dan Eropa (Irlandia).
ListDurableExecutionsByFunction Permintaan API	15 permintaan per detik.
SendDurableExecutionCallbackFailure Permintaan API	3.000 permintaan per detik. 6.000 permintaan per detik di AS Timur (Virginia Utara), AS Barat (Oregon), dan Eropa (Irlandia).
SendDurableExecutionCallbackHeartbeat Permintaan API	3.000 permintaan per detik. 6.000 permintaan per detik di AS Timur (Virginia Utara), AS Barat (Oregon), dan Eropa (Irlandia).

Sumber Daya	Kuota bawaan
SendDurableExecutionCallbackSuccess Permintaan API	3.000 permintaan per detik. 6.000 permintaan per detik di AS Timur (Virginia Utara), AS Barat (Oregon), dan Eropa (Irlandia).
StopDurableExecution Permintaan API	300 permintaan per detik. 600 permintaan per detik di AS Timur (Virginia Utara), AS Barat (Oregon), dan Eropa (Irlandia).

MicroVM Lambda

Lambda MicroVMS menetapkan kuota untuk permintaan komputasi, penyimpanan, dan API. Kuota yang ditandai sebagai dapat disesuaikan dapat ditingkatkan melalui konsol Kuota Layanan.

> **Note**
>
> Lambda MicroVMS mendukung ARM64 (AWS Arsitektur Graviton).

Komputasi dan penyimpanan

Sumber daya	Kuota bawaan	Dapat Disesuaikan
Memori dialokasikan di semua microVM (per akun, per Wilayah)	400 GB (yaitu, 200 microVM pada 2 GB memori yang dikonfigurasi masing-masing atau 400 microVM pada 1 GB memori yang dikonfigurasi masing-masing).	Ya

Sumber daya	Kuota bawaan	Dapat Disesuaikan
	1.024 GB di AS Timur (Virginia Utara), AS Barat (Oregon), AS Timur (Ohio), dan Asia Pasifik (Tokyo) (yaitu, 512 microVM dengan memori yang dikonfigurasi 2 GB masing-masing atau 1.024 microVM dengan memori yang dikonfigurasi 1 GB masing-masing). Meledak hingga empat kali kuota ini.	
Durasi eksekusi maksimum per microVM	8 jam (28,800 detik)	Tidak

Gambar dan versi

Sumber daya	Kuota bawaan	Dapat Disesuaikan
Gambar MicroVM per akun per Wilayah	100	Ya
Versi untuk gambar microVM	50	Ya
Pembuatan gambar bersamaan (per akun, per Wilayah)	5 10 di AS Timur (Virginia Utara), AS Barat (Oregon), AS Timur (Ohio), dan Asia Pasifik (Tokyo)	Ya

Per-MicroVM throughput

Sumber daya	Kuota	Dapat Disesuaikan
Koneksi bersamaan per microVM	8 (1 vCPU), 16 (2 vCPU), 32 (4 vCPU), 64 (8 vCPU), 128 (16 vCPU)	Tidak
Permintaan per detik per microVM	40 (4 vCPU/8 GB), 160 (16 vCPU/32 GB)	Tidak

Batas tarif API

Operasi API	Nilai (TPS)	Burst	Dapat Disesuaikan
`RunMicrovm`	5	5	Ya
`ResumeMicrovm`	5	5	Ya
`SuspendMicrovm`	2	2	Ya
`TerminateMicrovm`	10	10	Ya
`GetMicrovm`	100	100	Ya
`CreateMicrovmAuthToken`	50	50	Ya
`CreateMicrovmShellAuthToken`	5	5	Ya

> **Note**
>
> TPS = transaksi per detik. Batas tarif ini adalah per akun, per Wilayah. Coba lagi permintaan yang dibatasi menggunakan backoff eksponensial dengan jitter.

Fungsi Web Lambda

Dengan AWS Lambda Fungsi Web, Anda menjalankan aplikasi web dan backend API sebagai server HTTP tanpa server. Kuota adalah nilai maksimum untuk sumber daya dan pengaturan Fungsi Web. Contohnya termasuk jumlah titik akhir per Fungsi Web, memori lingkungan eksekusi, dan permintaan per detik untuk akun Anda. Mengetahui mereka sebelum Anda menerapkan membantu Anda merancang aplikasi dan mengukur pengaturan penskalaan titik puncaknya agar sesuai. Sebagian besar kuota memiliki nilai default yang berfungsi untuk beban kerja biasa dan tidak dapat diubah. Anda dapat meminta kenaikan untuk kuota akun yang dapat disesuaikan, yang dapat bervariasi menurut akun.

Kuota Fungsi Web berlaku pada cakupan berikut:

- Per Fungsi Web — Titik akhir dan revisi.
- Per titik akhir — Lingkungan maksimum, batas tarif, dan jumlah revisi yang berbagi lalu lintas.
- Per revisi — Ukuran paket penerapan, ukuran variabel lingkungan, dan batas waktu permintaan.
- Per lingkungan eksekusi — vCPU, memori, permintaan bersamaan, `/tmp` ukuran direktori, deskriptor file, proses dan utas, dan batas waktu inisialisasi.
- Per permintaan — Minta ukuran header.
- Per akun, di setiap Wilayah AWS — Permintaan per detik dan vCPU Arm64.

> **Note**
>
> Untuk melihat nilai kuota akun yang berlaku untuk akun Anda di Wilayah, hubungi `GetWebAccountSettings`. Untuk memantau penggunaan terhadap kuota ini, lihat the section called “Penggunaan regional dan metrik pelambatan”.

Fungsi Web, revisi, dan kuota lingkungan eksekusi

Tabel berikut mencantumkan kuota untuk setiap Fungsi Web, revisi, lingkungan eksekusi, dan permintaan.

Sumber daya	Kuota bawaan	Dapat Disesuaikan
Titik akhir per Fungsi Web	10	Tidak

Sumber daya	Kuota bawaan	Dapat Disesuaikan
Revisi untuk Fungsi Web	50 (target pembersihan otomatis). Hitungannya untuk sementara dapat melebihi 50. Lihat the section called “Batas revisi dan pembersihan otomatis”.	Tidak
Ukuran paket penerapan	512 MB (tidak di-zip)	Tidak
Ukuran variabel lingkungan	32KB	Tidak
Minta ukuran header	128 KiB	Tidak
`/tmp` ukuran direktori	512 MB	Tidak
vCPU per lingkungan eksekusi	2	Tidak
Memori per lingkungan eksekusi	2 GB	Tidak
Permintaan bersamaan per lingkungan (`maxConcurrencyPerEnvironment`)	64 (kisaran 1—128). Lihat the section called “Batas konkurensi dinamis per lingkungan”.	Tidak
Permintaan waktu habis	30 detik (kisaran 3—900)	Tidak
Batas waktu inisialisasi (waktu bagi server Anda untuk mulai mendengarkan)	10 detik	Tidak
Deskriptor file per lingkungan eksekusi	1.024	Tidak
Proses dan utas per lingkungan eksekusi	1.024	Tidak

Kuota titik akhir

Tabel berikut mencantumkan kuota titik akhir.

Sumber daya	Kuota bawaan	Dapat Disesuaikan
Lingkungan maksimum per titik akhir (`scalingConfig.maxEnvironments`)	Tidak wajib. Tidak diatur secara default. Nilai yang valid 2—10.000. diberlakukan di setiap wilayah. Kuota akun vCPU dapat membatasi titik akhir ke lingkungan eksekusi yang lebih sedikit.	Tidak
Batas tarif per titik akhir (`throttleConfig.rateLimit`)	Tidak wajib. Tidak diatur secara default. Nilai yang valid adalah 0, 100—1.000 dalam langkah 100, dan 2.000—10.000 dalam langkah 1.000 permintaan per detik. Lambda memberlakukan batas ini secara terpisah di setiap Wilayah. Nilai 0 memblokir semua lalu lintas. Jika tidak disetel, kuota per titik akhir sebesar 10.000 permintaan per detik berlaku.	Tidak
Revisi maksimum per titik akhir (bobot revisi)	2	Tidak

Sumber daya	Kuota bawaan	Dapat Disesuaikan
Kisaran berat revisi	1–100	Tidak

Kuota akun

Tabel berikut mencantumkan kuota akun, yang berlaku di masing-masing Wilayah AWS.

Sumber daya	Kuota bawaan	Dapat Disesuaikan
Permintaan per detik per akun	100,000	Ya
Arm64 vCPU untuk lingkungan eksekusi, per akun, per Wilayah	2.000. Setiap lingkungan eksekusi menggunakan 2 vCPU.	Ya

Kendala jaringan

Batasan berikut berlaku untuk jaringan Fungsi Web.

Item	Kendala
Konektor jaringan keluar per revisi	0 atau 1
Referensi konektor	Version-qualified ARN
Akun konektor dan Wilayah	Sama seperti fungsi web
Jenis sumber daya komputasi konektor	Tepat satu; gunakan `WebFunction`
Mengubah konektor revisi	Buat revisi baru
Revisi dirutekan oleh satu titik akhir	1 atau 2
Jenis titik akhir yang mendukung konektor jaringan keluar	`HomeRegion` cuma

Item	Kendala
Penyebaran otomatis	`LatestRevision` atau `Disabled` untuk `HomeRegion` ; `Disabled` hanya untuk `MultiRegion` dan `PerRegion`
Jenis titik akhir dan Set Wilayah	Tetap pada pembuatan titik akhir
Keluar publik langsung	`IPv4` cuma
Akses endpoint pribadi dengan AWS PrivateLink	Tidak tersedia

Layanan lainnya

Kuota untuk layanan lain, seperti AWS Identity and Access Management (IAM), Amazon CloudFront (Lambda @Edge), dan Amazon Virtual Private Cloud (Amazon VPC), dapat memengaruhi fungsi Lambda Anda. Untuk informasi lebih lanjut, lihat Layanan AWS kuota di Referensi Umum Amazon Web Services, dan Memanggil Lambda dengan acara dari yang lain AWS layanan.

Banyak aplikasi yang melibatkan Lambda menggunakan beberapa AWS layanan. Karena layanan yang berbeda memiliki kuota yang berbeda untuk berbagai fitur, mungkin sulit untuk mengelola kuota ini di seluruh aplikasi Anda. Misalnya, API Gateway memiliki batas throttle default 10.000 permintaan per detik, sedangkan Lambda memiliki batas konkurensi default 1.000. Karena ketidakcocokan ini, dimungkinkan untuk memiliki lebih banyak permintaan masuk dari API Gateway yang dapat ditangani Lambda. Anda dapat mengatasinya dengan meminta kenaikan batas konkurensi Lambda agar sesuai dengan tingkat lalu lintas yang diharapkan.

Pengujian beban aplikasi Anda membantu Anda memantau kinerja aplikasi secara menyeluruh sebelum menerapkan ke produksi. Selama uji beban, Anda dapat mengidentifikasi kuota apa pun yang dapat bertindak sebagai faktor pembatas untuk tingkat lalu lintas yang Anda harapkan dan mengambil tindakan yang sesuai.

Riwayat dokumen

Tabel berikut menjelaskan perubahan penting pada Panduan Developer AWS Lambda sejak Mei 2018. Untuk notifikasi tentang pembaruan dokumentasi ini, langganan ke umpan RSS.

Perubahan	Deskripsi	Tanggal
AWS pembaruan kebijakan terkelola	Lambda menambahkan kebijakan terkel AWS ola baru (`AWSLambdaNetworkConnectorOperatorPolicy`) untuk memberikan izin untuk membuat dan mengelola sumber daya antarmuka jaringan elastis (ENI) yang dikelola oleh Konektor Jaringan Lambda. Untuk informasi selengkapnya, lihat Pembaruan Lambda untuk kebijakan ter AWS kelola.	Juni 19, 2026
AWS pembaruan kebijakan terkelola	Lambda memperbarui kebijakan `AWSLambdaManagedEC2ResourceOperator` terkelola untuk menambahkan CloudWatch izin Amazon (`logs:CreateLogGroup`, `logs:CreateLogStream`, `logs:PutLogEvents`). Untuk detailnya, lihat Pembaruan Lambda untuk kebijakan ter AWS kelola.	Juni 10, 2026
Self-managed Penyimpanan kode S3	Lambda sekarang mendukung bucket S3 yang dikelola	30 April 2026

	sendiri untuk fungsi dan penyimpanan kode lapisan. Anda dapat mengonfigurasi Lambda untuk mereferen sikan kode Anda langsung dari bucket S3 Anda, menghilan gkan batas Lambda-managed penyimpanan. Untuk detailnya , lihat penyimpanan kode Self-managed S3.	
AWS pembaruan kebijakan terkelola	Lambda memperbarui kebijakan `AWSLambda ManagedEC2Resource Operator` terkelola untuk menambahkan `ec2:Descr ibeVpcEncryptionCo ntrols` izin. Untuk detailnya , lihat Pembaruan Lambda untuk kebijakan ter AWS kelola.	April 23, 2026
AWS pembaruan kebijakan terkelola	Lambda telah merilis kebijakan terkelola baru `AWSLambda BasicDurableExecut ionRolePolicy` yang memberikan izin menulis ke CloudWatch Log dan read/writ e izin ke API eksekusi tahan lama yang digunakan oleh fungsi tahan lama Lambda.	Desember 1, 2025

Service-linked peran untuk Lambda	Lambda sekarang mendukung role terkait layanan (`AWSServiceRoleForLambda`) yang memungkinkan Lambda untuk menghentikan instans yang dikelola sebagai bagian dari penyedia kapasitas Lambda.	30 November 2025
Instans Terkelola Lambda	Lambda sekarang mendukung Instans Terkelola, yang menyediakan kapasitas komputasi khusus untuk fungsi Anda dengan kinerja, keamanan, dan kontrol yang ditingkatkan. Untuk detailnya, lihat Instans Terkelola Lambda.	30 November 2025
AWS pembaruan kebijakan terkelola	Lambda menambahkan kebijakan terkel AWS ola baru (`AWSLambdaBasicDurableExecutionRolePolicy`).	30 November 2025
AWS pembaruan kebijakan terkelola	Lambda memperbarui kebijakan ter AWS kelola yang ada`AWSLambda_FullAccess`) memungkinkan `iam:CreateServiceLinkedRole` tindakan `kms:DescribeKey` dan.	30 November 2025

AWS pembaruan kebijakan terkelola	Lambda menambahkan kebijakan terkel AWS ola baru (`AWSLambdaManagedEC2ResourceOperator`) untuk mengaktifkan manajemen instans Amazon EC2 otomatis untuk penyedia kapasitas Lambda. Untuk detailnya, lihat Pembaruan Lambda untuk kebijakan ter AWS kelola.	30 November 2025
AWS pembaruan kebijakan terkelola	Lambda menambahkan kebijakan terkel AWS ola baru (`AWSLambdaServiceRolePolicy`). Untuk detailnya, lihat Pembaruan Lambda untuk kebijakan ter AWS kelola.	30 November 2025
AWS pembaruan kebijakan terkelola	Lambda memperbarui dua kebijakan terkel AWS ola yang ada (`AWSLambda_ReadOnlyAccess` dan`AWSLambda_FullAccess`).	Februari 20, 2025
Node.js Waktu pelaksanaan 22.x	Lambda sekarang mendukung Node.js 22 sebagai runtime terkelola dan image basis kontainer (`nodejs22.x`).	November 22, 2024
SnapStart dukungan untuk Python dan .NET	Lambda sekarang SnapStart tersedia untuk runtime terkelola Python dan .NET, dimulai dengan `python3.12` dan. `dotnet8`	November 18, 2024

Waktu proses Python 3.13	Lambda sekarang mendukung Python 3.13 sebagai runtime terkelola dan image basis kontainer.	November 13, 2024
Enkripsi yang dikelola pelanggan untuk paket penyebaran.zip	Lambda sekarang mendukung enkripsi kunci AWS KMS yang dikelola pelanggan untuk paket penyebaran.zip.	8 November 2024
Dukungan untuk SnapStart di Wilayah baru	Lambda sekarang SnapStart tersedia di Wilayah berikut: Eropa (Spanyol), Eropa (Zurich), Asia Pasifik (Melbourne), Asia Pasifik (Hyderabad), dan Timur Tengah (UEA).	Januari 12, 2024
AWS pembaruan kebijakan terkelola	Lambda memperbarui kebijakan terkel AWS ola yang ada (`AWSLambdaVPCAccess ExecutionRole`).	Januari 5, 2024
Waktu proses Python 3.12	Lambda sekarang mendukung Python 3.12 sebagai runtime terkelola dan image basis kontainer. Untuk informasi lebih lanjut, lihat runtime Python 3.12 sekarang tersedia AWS Lambda di AWS Compute Blog.	14 Desember 2023
Waktu pelaksanaan Java 21	Lambda sekarang mendukung Java 21 sebagai runtime terkelola dan image basis container (`java21`).	16 November 2023

Node.js Waktu pelaksanaan 20.x	Lambda sekarang mendukung Node.js 20 sebagai runtime terkelola dan image basis kontainer (`nodejs20.x`). Untuk informasi selengkapnya, lihat runtime Node.js 20.x sekarang tersedia AWS Lambda di AWS Compute Blog.	14 November 2023
sediakan.al2023 waktu pelaksanaan.al2023	Lambda sekarang mendukung Amazon Linux 2023 sebagai runtime terkelola dan image basis kontainer. Untuk informasi selengkapnya, lihat Memperkenalkan runtime Amazon Linux 2023 untuk AWS Lambda di AWS Compute Blog.	9 November 2023
Dukungan IPv6 untuk subnet dual-stack	Lambda sekarang mendukung lalu lintas IPv6 keluar ke subnet dual-stack. Untuk informasi selengkapnya, lihat dukungan https://docs.aws.amazon.com/lambda/latest/dg/foundation-networking.html?icmpid=docs_lambda_rss#foundation-nw-ipv6 IPv6.	12 Oktober 2023

Menguji fungsi dan aplikasi tanpa server	Pelajari tentang teknik untuk men-debug dan mengotoma tiskan pengujian fungsi tanpa server di cloud. Sekarang ada bagian pengujian dan sumber daya yang disertakan dalam bagian bahasa Python dan TypeScript. Untuk detailnya , lihat Menguji fungsi dan aplikasi tanpa server.	Juni 16, 2023
Runtime Ruby 3.2	Lambda sekarang mendukung runtime baru untuk Ruby 3.2. Untuk informasi selengkap nya, lihat Membangun fungsi Lambda dengan Ruby.	Juni 7, 2023
Streaming tanggapan	Lambda sekarang mendukung respons streaming dari fungsi. Untuk informasi selengkapnya, lihat Meng onfigurasi fungsi Lambda untuk mengalirkan respons.	6 April 2023
Metrik pemanggilan asinkron	Lambda merilis metrik pemanggilan asinkron. Untuk informasi selengkapnya, lihat Metrik pemanggilan asinkron.	9 Februari 2023

Kontrol versi runtime	Lambda merilis versi runtime baru yang mencakup pembaruan keamanan, perbaikan bug, dan fitur baru. Anda sekarang dapat mengontrol kapan fungsi Anda diperbarui ke versi runtime baru. Untuk informasi selengkapnya, lihat Pembaruan runtime Lambda.	23 Januari 2023
Lambda SnapStart	Gunakan Lambda SnapStart untuk mengurangi waktu startup untuk fungsi Java tanpa menyediakan sumber daya tambahan atau menerapkan pengoptimalan kinerja yang kompleks. Untuk informasi selengkapnya, lihat Meningkatkan kinerja startup dengan menggunakan Lambda SnapStart.	28 November 2022
Node.js 18 periode waktu pelaksanaan	Lambda sekarang mendukung runtime baru untuk Node.js 18. Node.js 18 menggunakan Amazon Linux 2. Untuk detailnya, lihat Membangun fungsi Lambda dengan Node.js.	18 November 2022

lambda: kunci SourceFunctionArn kondisi	Untuk AWS sumber daya, kunci `lambda:SourceFunctionArn` kondisi menyaring akses ke sumber daya oleh ARN dari fungsi Lambda. Untuk detailnya, lihat Bek erja dengan kre dentif lingkungan eksekusi Lambda.	1 Juli 2022
Node.js 16 periode waktu pelaksanaan	Lambda sekarang mendukung runtime baru untuk Node.js 16. Node.js 16 menggunakan Amazon Linux 2. Untuk detailnya, lihat Mem bangun fungsi Lambda dengan Node.js.	Mei 11, 2022
URL fungsi lambda	Lambda sekarang mendukung URL fungsi, yang merupakan titik akhir HTTP (S) khusus untuk fungsi Lambda. Untuk detailnya, lihat URL fungsi Lambda.	April 6, 2022
Acara pengujian bersama di AWS Lambda konsol	Lambda sekarang mendukung berbagi acara pengujian dengan pengguna lain dalam hal yang sama Akun AWS. Untuk detailnya, lihat Menguji fungsi Lambda di konsol.	16 Maret 2022
PrincipalOrgId dalam kebijakan berbasis sumber daya	Lambda sekarang mendukung pemberian izin ke organisasi di AWS Organizations. Untuk detailnya, lihat Menggunakan kebijakan berbasis sumber daya untuk. AWS Lambda	11 Maret 2022

Waktu pelaksanaan.NET 6	Lambda sekarang mendukung runtime baru untuk.NET 6. Untuk detailnya, lihat Runtime Lambda.	Februari 23, 2022
Pemfilteran peristiwa untuk sumber peristiwa Kinesis, DynamoDB, dan Amazon SQS	Lambda sekarang mendukung pemfilteran peristiwa untuk sumber peristiwa Kinesis, DynamoDB, dan Amazon SQS. Untuk detailnya, lihat Pemfilter https://docs.aws.amazon.com/lambda/latest/dg/invocation-eventfiltering.html?icmpid=docs_lambda_rss an peristiwa Lambda.	24 November 2021
Otentikasi MTL untuk Amazon MSK dan sumber peristiwa Apache Kafka yang dikelola sendiri	Lambda sekarang mendukung otentikasi MTL untuk Amazon MSK dan sumber peristiwa Apache Kafka yang dikelola sendiri. Untuk detailnya, lihat Menggunakan Lambda dengan Amazon MSK.	November 19, 2021
Lambda pada Graviton2	Lambda sekarang mendukung Graviton2 untuk fungsi menggunakan arsitektu r arm64. Untuk detailnya, lihat Arsitektur set instruksi Lambda.	29 September 2021
Waktu proses Python 3.9	Lambda sekarang mendukung runtime baru untuk Python 3.9. Untuk detailnya, lihat Runtime Lambda.	16 Agustus 2021

Versi runtime baru untuk Node.js, Python, dan Java	Versi runtime baru tersedia untuk Node.js, Python, dan Java. Untuk detailnya, lihat Runtime Lambda.	21 Juli 2021
Dukungan untuk RabbitMQ sebagai sumber acara di Lambda	Lambda sekarang mendukung Amazon MQ untuk RabbitMQ sebagai sumber acara. Amazon MQ adalah layanan broker pesan terkelola untuk Apache ActiveMQ dan RabbitMQ yang memudahkan penyiapan dan pengoperasian broker pesan di cloud. Untuk detailnya, lihat Menggunakan Lambda dengan Amazon MQ.	7 Juli 2021
SASL/PLAIN otentikasi untuk Kafka yang dikelola sendiri di Lambda	SASL/PLAIN sekarang merupakan mekanisme otentikasi yang didukung untuk sumber peristiwa Kafka yang dikelola sendiri di Lambda Pelanggan yang sudah digunakan SASL/PLAIN pada klaster Kafka yang dikelola sendiri sekarang dapat dengan mudah menggunakan Lambda untuk membangun aplikasi konsumen tanpa harus memodifikasi cara mereka mengautentikasi. Untuk detailnya, lihat Menggunakan Lambda dengan Apache Kafka yang dikelola sendiri.	29 Juni 2021

API Ekstensi Lambda	Ketersediaan umum untuk ekstensi Lambda. Gunakan ekstensi untuk menambah fungsi Lambda Anda. Anda dapat menggunakan ekstensi yang disediakan oleh Partner Lambda, atau Anda dapat membuat ekstensi Lambda Anda sendiri. Untuk detailnya, lihat API Ekstensi Lambda.	24 Mei 2021
Pengalaman konsol Lambda baru	Konsol Lambda telah didesain ulang untuk meningkatkan performa dan konsistensi.	2 Maret 2021
Node.js 14 periode waktu pelaksanaan	Lambda sekarang mendukung runtime baru untuk Node.js 14. Node.js 14 menggunakan Amazon Linux 2. Untuk detailnya, lihat Membangun fungsi Lambda dengan Node.js.	27 Januari 2021
Gambar wadah lambda	Lambda sekarang mendukung fungsi yang didefinisikan sebagai gambar kontainer. Anda dapat menggabungkan fleksibilitas peralatan kontainer dengan ketangkasan dan kesederhanaan operasional Lambda untuk membangun aplikasi. Untuk detail selengkapnya, lihat Menggunakan gambar kontainer dengan Lambda.	1 Desember 2020

Penandatanganan kode untuk fungsi Lambda	Lambda kini mendukung penandatanganan kode. Administrator dapat mengonfigurasi fungsi Lambda untuk hanya menerima kode yang ditandatangani pada deployment. Lambda memeriksa tanda tangan untuk memastikan kode tidak diubah atau disusupi. Selain itu, Lambda memastikan kode sudah ditandatangani oleh developer tepercaya sebelum menerima deployment. Untuk detailnya, lihat Mengonfigurasi penandatanganan kode untuk Lambda.	23 November 2020
Pratinjau: API Lambda Runtime Logs	Lambda kini mendukung API Log Runtime. Ekstensi Lambda dapat menggunakan API Log untuk mengikuti aliran log di lingkungan pelaksanaan. Untuk detailnya, lihat API Log Runtime Lambda.	12 November 2020
Sumber acara baru untuk Amazon MQ	Lambda kini mendukung Amazon MQ sebagai sumber peristiwa. Gunakan fungsi Lambda untuk memproses catatan dari broker pesan Amazon MQ Anda. Untuk detailnya, lihat Menggunakan Lambda dengan Amazon MQ.	5 November 2020

Pratinjau: API Ekstensi Lambda	Gunakan ekstensi Lambda untuk menambah fungsi Lambda Anda. Anda dapat menggunakan ekstensi yang disediakan oleh Lambda Partners, atau Anda dapat membuat ekstensi Lambda Anda sendiri. Untuk detailnya, lihat API Ekstensi Lambda.	8 Oktober 2020
Dukungan untuk Java 8 dan runtime khusus pada AL2	Lambda kini mendukung Java 8 dan runtime kustom di Amazon Linux 2. Untuk detailnya, lihat Runtime Lambda.	12 Agustus 2020
Sumber acara baru untuk Amazon Managed Streaming untuk Apache Kafka	Lambda kini mendukung Amazon MSK sebagai sumber peristiwa. Gunakan fungsi Lambda dengan Amazon MSK untuk memproses catatan dalam topik Kafka. Untuk detailnya, lihat Menggunakan Lambda dengan Amazon MSK.	11 Agustus 2020

Kunci kondisi IAM untuk pengaturan Amazon VPC	Anda sekarang dapat menggunakan kunci Lambda-specific kondisi untuk pengaturan VPC. Misalnya, Anda dapat meminta semua fungsi dalam organisasi Anda terhubung ke VPC. Anda juga dapat menentukan subnet dan grup keamanan yang dapat dan tidak dapat digunakan oleh pengguna fungsi. Untuk detailnya, lihat Mengonfigurasi VPC untuk fungsi IAM.	10 Agustus 2020
Pengaturan konkurensi untuk konsumen aliran Kinesis HTTP/2	Anda sekarang dapat menggunakan pengatura n konkurensi berikut untuk konsumen Kinesis dengan fan-out (HTTP/2 aliran) yang disempurnakan: Paralleli zationFactor,,, MaximumRe tryAttempts, MaximumRe cordAgeInSeconds dan. DestinationConfig BisectBat chOnFunctionError Untuk detailnya, lihat Menggunakan AWS Lambda dengan Amazon Kinesis.	7 Juli 2020

Jendela batch untuk konsumen HTTP/2 streaming Kinesis	Anda sekarang dapat mengonfigurasi jendela batch (MaximumBatchingWindowInSeconds) untuk HTTP/2 aliran. Lambda membaca catatan dari pengaliran hingga memenuhi batch, atau hingga rentang waktu batch kedaluwarsa. Untuk detailnya, lihat Menggunakan AWS Lambda dengan Amazon Kinesis.	Selasa, 18 Juni 2020
Dukungan untuk sistem file Amazon EFS	Sekarang Anda dapat menghubungkan sistem file Amazon EFS ke fungsi Lambda Anda untuk akses file jaringan bersama. Untuk detailnya, lihat Mengonfigurasi akses sistem file untuk fungsi Lambda.	16 Juni 2020
AWS CDK contoh aplikasi di konsol Lambda	Konsol Lambda sekarang menyertakan contoh aplikasi yang menggunakan AWS Cloud Development Kit (AWS CDK) for TypeScript. Ini AWS CDK adalah kerangka kerja yang memungkinkan Anda untuk menentukan sumber daya aplikasi Anda di TypeScript, Python, Java, atau.NET.	1 Juni 2020

Dukungan untuk runtime .NET Core 3.1.0 di AWS Lambda	AWS Lambda sekarang mendukung runtime .NET Core 3.1.0. Untuk detailnya, lihat .NET Core CLI.	31 Maret 2020
Dukungan untuk API Gateway HTTP API	Dokumentasi yang diperbarui dan diperluas untuk menggunakan Lambda dengan API Gateway, termasuk dukungan untuk API HTTP. Menambahkan contoh aplikasi yang membuat API dan fungsi dengan CloudFormation. Untuk detailnya, lihat Menggunakan Lambda dengan Amazon API Gateway.	23 Maret 2020
Ruby 2.7	Runtime baru tersedia untuk Ruby 2.7, ruby2.7, yang merupakan runtime Ruby pertama yang menggunakan Amazon Linux 2. Untuk detailnya, lihat Membangun fungsi Lambda dengan Ruby.	19 Februari 2020

Metrik konkurensi	Lambda kini melaporkan metrik `ConcurrentExecutions` untuk semua fungsi, alias, dan versi. Anda bisa melihat grafik untuk metrik ini pada halaman pemantauan untuk fungsi Anda. Sebelumnya, `ConcurrentExecutions` hanya dilaporkan pada tingkat akun dan untuk fungsi yang menggunakan konkurensi yang telah ditentukan. Untuk detailnya, lihat Metrik fungsi AWS Lambda.	18 Februari 2020

Perbarui ke status fungsi	Status fungsi kini diterapkan untuk semua fungsi secara default. Saat Anda menghubungkan fungsi ke VPC, Lambda membuat antarmuka jaringan elastis bersama. Hal ini memungkinkan peningkatan skala fungsi Anda tanpa membuat antarmuka jaringan tambahan. Selama proses ini, Anda tidak dapat melakukan operasi tambahan pada fungsi, termasuk memperbarui konfigurasinya dan menerbitkan versi. Dalam beberapa kasus, invokasi juga terdampak. Detail tentang status fungsi saat ini tersedia dari API Lambda. Pembaruan ini dirilis secara bertahap. Untuk detailnya, lihat siklus hidup status Lambda yang diperbarui untuk jaringan VPC di Blog AWS Komputasi. Untuk informasi selengkapnya tentang status, lihat status fungsi AWS Lambda.	24 Januari 2020

Pembaruan untuk output API konfigurasi fungsi	Menambahkan kode alasan ke StateReasonCode (InvalidSubnet, InvalidSecurityGroup) dan LastUpdateStatusReasonCode (SubnetOutOfIPAddresses, InvalidSubnet, InvalidSecurityGroup) untuk fungsi yang terhubung ke VPC. Untuk informasi selengkapnya tentang status, lihat status fungsi AWS Lambda.	20 Januari 2020
Konkurensi yang disediakan	Anda dapat mengalokasikan konkurensi terprovisi untuk versi fungsi, atau alias. Konkurensi terprovisi memungkinkan fungsi menskalakan tanpa fluktuasi di latensi. Untuk detailnya, lihat Mengelola konkurensi untuk fungsi Lambda.	3 Desember 2019
Buat proxy database	Anda dapat menggunakan konsol Lambda untuk membuat proxy basis data untuk fungsi Lambda. Proksi basis data memungkinkan fungsi mencapai tingkat konkurensi tinggi tanpa membuang koneksi basis data. Untuk detailnya, lihat Mengonfigurasi akses basis data untuk fungsi Lambda.	3 Desember 2019

Dukungan persentil untuk metrik durasi	Anda dapat memfilter metrik durasi berdasarkan persentil . Untuk detailnya, lihat Metrik AWS Lambda.	26 November 2019
Peningkatan konkurensi untuk sumber acara streaming	Opsi baru untuk pemetaan sumber peristiwa DynamoDB stream dan Kinesis stream memungkinkan Anda memproses lebih dari satu batch sekaligus dari setiap serpihan. Saat Anda meningkatkan jumlah batch bersamaan per serpihan, konkurensi fungsi Anda dapat mencapai 10 kali jumlah serpihan di pengaliran Anda. Untuk detailnya, lihat Pemetaan sumber peristiwa Lambda.	25 November 2019

Status fungsi	Saat Anda membuat atau memperbarui fungsi, fungsi memasuki status tertunda sewaktu Lambda menyediakan sumber daya untuk mendukungnya. Jika Anda menghubungkan fungsi Anda ke VPC, Lambda dapat langsung membuat antarmuka jaringan elastis bersama, alih-alih membuat antarmuka jaringan saat fungsi Anda diminta. Ini menghasilkan kinerja yang lebih baik untuk VPC-connected fungsi, tetapi mungkin memerlukan pembaruan untuk otomatisasi Anda. Untuk detailnya, lihat status fungsi AWS Lambda.	25 November 2019
Opsi penanganan kesalahan untuk pemanggilan asinkron	Opsi konfigurasi baru sudah tersedia untuk invokasi asinkron. Anda dapat mengonfigurasi Lambda untuk membatasi percobaan ulang dan mengatur periode peristiwa maksimum. Untuk detailnya, lihat Mengonfigurasi penanganan kesalahan untuk invokasi asinkron.	25 November 2019

Penanganan kesalahan untuk sumber peristiwa aliran	Opsi konfigurasi baru tersedia untuk pemetaan sumber peristiwa yang membaca dari pengaliran. Anda dapat mengonfigurasi pemetaan sumber peristiwa DynamoDB stream dan Kinesis stream untuk membatasi percobaan ulang dan menetapkan periode catatan maksimum. Jika terjadi kesalahan, Anda dapat mengonfigurasi pemetaan sumber peristiwa untuk membagikan batch sebelum mencoba kembali, dan untuk mengirim catatan invokasi bagi batch yang gagal pada antrean atau topik. Untuk detailnya, lihat Pemetaan sumber peristiwa Lambda.	25 November 2019
Tujuan untuk pemanggilan asinkron	Anda dapat mengonfigurasi Lambda untuk mengirim rekaman invokasi asinkron ke layanan lain. Catatan invokasi berisi detail tentang peristiwa , konteks, dan respons fungsi. Anda dapat mengirim catatan pemanggilan ke antrian SQS, topik SNS, fungsi Lambda, atau bus acara. EventBrid ge Untuk detailnya, lihat Mengonfigurasi tujuan untuk invokasi asinkron.	25 November 2019

Runtime baru untuk Node.js, Python, dan Java	Runtime baru tersedia untuk Node.js 12, Python 3.8, dan Java 11. Untuk detailnya, lihat Runtime Lambda.	18 November 2019
Dukungan antrian FIFO untuk sumber acara Amazon SQS	Anda dapat membuat pemetaan sumber peristiwa yang membaca dari antrean masuk pertama dan keluar pertama (first-in, first-out [FIFO]). Sebelumnya, hanya antrean standar yang didukung. Untuk detailnya, lihat Menggunakan Lambda dengan Amazon SQS.	18 November 2019
Buat aplikasi di konsol Lambda	Pembuatan aplikasi di konsol Lambda sekarang tersedia secara umum. Untuk petunjuk, lihat M engelola aplikasi di konsol Lambda.	31 Oktober 2019

Buat aplikasi di konsol Lambda (beta)	Anda dapat membuat aplikasi Lambda yang dengan alur pengiriman berkelanjutan terpadu di konsol Lambda. Konsol menyediakan aplikasi sampel yang dapat Anda gunakan sebagai titik awal untuk proyek Anda. Pilih antara AWS CodeCommit dan GitHub untuk kontrol sumber. Setiap kali Anda melakukan perubahan pada repositori Anda, alur yang disertakan membangun dan men-deplo y secara otomatis. Untuk petunjuk, lihat M engelola aplikasi di konsol Lambda.	3 Oktober 2019

Peningkatan kinerja untuk VPC-connected fungsi	Lambda sekarang menggunakan antarmuka jaringan elastis jenis baru yang dipakai bersama oleh semua fungsi dalam subnet virtual private cloud (VPC) Ketika Anda menghubungkan fungsi ke VPC, Lambda membuat antarmuka jaringan untuk setiap kombinasi kelompok keamanan dan subnet yang Anda pilih. Ketika antarmuka jaringan bersama tersedia, fungsi tidak perlu lagi membuat antarmuka jaringan tambahan saat jaringan menaikkan skala. Hal ini sangat meningkatkan waktu startup. Untuk detail, lihat Mengkonfigurasi Fungsi Lambda untuk mengakses sumber daya di VPC.	Selasa, 03 September 2019
Pengaturan batch streaming	Sekarang Anda dapat mengonfigurasi jendela batch untuk sumber peristiwa pemetaan Amazon DynamoDB dan Amazon Kinesis. Konfigurasikan jendela batch hingga lima menit untuk buffer catatan yang baru hingga batch penuh tersedia. Ini mengurangi berapa kali fungsi Anda diaktifkan saat pengaliran kurang aktif.	Selasa, 29 Agustus 2019

CloudWatch Integrasi Logs Insights	Halaman pemantauan di konsol Lambda sekarang menyertakan laporan dari Amazon CloudWatch Logs Insights.	Selasa, 18 Juni 2019
Amazon Linux 2018.03	Lingkungan eksekusi Lambda sedang diperbarui untuk menggunakan Amazon Linux 2018.03. Untuk detailnya, lihat Lingkungan eksekusi.	21 Mei 2019
Node.js 10	Runtime baru tersedia untuk Node.js 10, nodejs10.x. Runtime ini menggunakan Node.js 10.15 dan akan diperbarui dengan rilis poin terbaru dari Node.js 10 secara berkala. Node.js 10 juga merupakan runtime pertama yang menggunakan Amazon Linux 2. Untuk detailnya, lihat Mem bangun fungsi Lambda dengan Node.js.	Selasa, 13 Mei 2019
GetLayerVersionByArn API	Gunakan GetLayerVersionByArn API untuk mengunduh informasi versi layer dengan versi ARN sebagai input. Dibandingkan dengan GetLayerVersion, GetLayerVersionByArn memungkinkan Anda menggunakan ARN secara langsung alih-alih menguraikannya untuk mendapatkan nama layer dan nomor versi.	Selasa, 25 April 2019

Ruby	AWS Lambda sekarang mendukung Ruby 2.5 dengan runtime baru. Untuk detailnya, lihat Membangun fungsi Lambda dengan Ruby.	Selasa, 29 Nopember 2018
Lapisan	Dengan layer Lambda, Anda dapat mengemas dan menyebarkan pustaka, Runtime khusus, dan ketergantungan lain secara terpisah dari kode fungsi Anda. Bagikan lapisan Anda dengan akun lain atau seluruh dunia. Untuk detailnya, lihat layer Lambda.	29 November 2018
Waktu proses kustom	Bangun runtime khusus untuk menjalankan fungsi Lambda pada bahasa pemrogram an yang Anda sukai. Untuk detailnya, lihat Runtime Lambda khusus.	29 November 2018
Pemicu Penyeimbang Beban Aplikasi	Elastic Load Balancing kini mendukung fungsi Lambda sebagai target untuk Applicati on Load Balancer. Untuk detailnya, lihat Menggunak an Lambda Application Load Balancer.	29 November 2018

Gunakan konsumen HTTP/2 aliran Kinesis sebagai pemicu	Anda dapat menggunakan konsumen aliran HTTP/2 data Kinesis untuk mengirim acara ke AWS Lambda. Konsumen streaming memiliki throughput baca khusus dari setiap pecahan dalam aliran data Anda dan digunakan HTTP/2 untuk meminimal kan latensi. Untuk detailnya, lihat Menggunakan Lambda dengan Kinesis.	19 November 2018
Python 3.7	AWS Lambda sekarang mendukung Python 3.7 dengan runtime baru. Untuk informasi selengkapnya, lihat Membangun fungsi Lambda dengan Python.	Selasa, 19 Nopember 2018
Peningkatan batas muatan untuk pemanggilan fungsi asinkron	Ukuran payload maksimum untuk invokasi asinkron naik dari 128 KB menjadi 256 KB, yang sesuai dengan ukuran pesan maksimum dari pemicu Amazon SNS. Untuk detailnya, lihat Kuota Lambda.	16 November 2018
AWS GovCloud (US-East) Wilayah	AWS Lambda sekarang tersedia di Wil AWS GovCloud ayah (US-East).	12 November 2018

Memindahkan AWS SAM topik ke Panduan Pengembang terpisah	Sejumlah topik difokuskan pada pembuatan aplikasi tanpa server menggunakan AWS Serverless Application Model (AWS SAM). Topik ini telah dipindahkan ke panduan AWS Serverless Application Model pengembang.	25 Oktober 2018
Lihat aplikasi Lambda di konsol	Anda dapat melihat status aplikasi Lambda Anda di halaman Aplikasi di konsol Lambda. Halaman ini menunjukkan status tumpu CloudFormation kan. Ini mencakup tautan ke halaman tempat Anda dapat melihat informasi selengkapnya tentang sumber daya dalam tumpukan. Anda juga dapat melihat metrik agregat untuk aplikasi dan membuat dasbor pemantauan kustom.	11 Oktober 2018
Batas waktu tunggu eksekusi fungsi	Untuk memungkinkan fungsi yang berlangsung lama, waktu habis eksekusi maksimum yang dapat dikonfigurasi akan meningkat dari 5 menit menjadi 15 menit. Untuk detailnya, lihat batas Lambda.	10 Oktober 2018

Dukungan untuk bahasa PowerShell Inti di AWS Lambda	AWS Lambda sekarang mendukung bahasa PowerShell inti. Untuk informasi selengkapnya, lihat Model pemrograman untuk penulisan fungsi Lambda di. PowerShell	11 September 2018
Dukungan untuk runtime .NET Core 2.1.0 di AWS Lambda	AWS Lambda sekarang mendukung runtime .NET Core 2.1.0. Untuk informasi selengkapnya, lihat .NET Core CLI.	9 Juli 2018
Pembaruan kini tersedia melalui RSS	Sekarang Anda dapat berlangganan ke umpan RSS untuk mengikuti rilis untuk panduan ini.	5 Juli 2018
Dukungan untuk Amazon SQS sebagai sumber acara	AWS Lambda sekarang mendukung Amazon Simple Queue Service (Amazon SQS) sebagai sumber acara. Untuk informasi lebih lanjut, lihat Memanggil fungsi Lambda.	28 Juni 2018
Wilayah China (Ningxia)	AWS Lambda sekarang tersedia di Wilayah China (Ningxia). Untuk informasi selengkapnya tentang Wilayah Lambda dan titik akhir, lihat Wil ayah dan titik akhir di. Referensi Umum AWS	28 Juni 2018

Pembaruan sebelumnya

Tabel berikut menjelaskan perubahan penting dalam setiap rilis Panduan Developer AWS Lambda sebelum Juni 2018.

Perubahan	Deskripsi	Date
Dukungan runtime untuk Node.js runtime 8.10	AWS Lambda sekarang mendukung Node.js runtime versi 8.10. Untuk informasi selengkapnya, lihat Membangun Fungsi Acara Lambda dengan Node.js.	2 April 2018
ID revisi fungsi dan alias	AWS Lambda sekarang mendukung ID revisi pada versi fungsi dan alias Anda. Anda dapat menggunakan ID ini untuk melacak dan menerapkan pembaruan kondisional saat Anda memperbarui versi fungsi atau sumber daya alias Anda.	25 Januari 2018
Dukungan runtime untuk Go dan .NET 2.0	AWS Lambda telah menambahkan dukungan runtime untuk Go dan .NET 2.0. Untuk informasi selengkapnya, lihat Membangun Fungsi Acara Lambda dengan Go dan Membangun Fungsi Acara Lambda dengan C #.	Selasa, 15 Januari 2018
Desain Ulang Konsol	AWS Lambda telah memperkenalkan konsol Lambda baru untuk menyederhanakan pengalaman Anda dan menambahkan Editor Kode Cloud9 untuk meningkatkan kemampuan Anda men-debug dan merevisi kode fungsi Anda.	30 November 2017
Mengatur Batas Konkurensi pada Fungsi Individu	AWS Lambda sekarang mendukung pengaturan batas konkurensi pada fungsi individu. Untuk informasi selengkapnya, lihat Mengkonfigurasi konkurensi cadangan untuk Fungsi Acara.	30 November 2017
Pergeseran Lalu Lintas dengan Alias	AWS Lambda sekarang mendukung pergeseran lalu lintas dengan alias. Untuk informasi selengkapnya, lihat Buat penerapan bergulir dengan alias tertimbang.	Selasa, 28 Nopember 2017

Perubahan	Deskripsi	Date
Deployment Kode Gradual	AWS Lambda sekarang mendukung penerapan versi baru fungsi Lambda Anda dengan aman menggunakan Code Deploy. Untuk informasi selengkapnya, lihat Deployment kode gradual.	28 November 2017
Wilayah Tiongkok (Beijing)	AWS Lambda sekarang tersedia di Wilayah China (Beijing). Untuk informasi selengkapnya tentang wilayah Lambda dan titik akhir, lihat Wil ayah dan titik akhir di. Referensi Umum AWS	Selasa, 09 Nopember 2017
Memperkenalkan SAM Local	AWS Lambda memperkenalkan SAM Local (sekarang dikenal sebagai SAM CLI), AWS CLI alat yang menyediak an lingkungan bagi Anda untuk mengembangkan, menguji, dan menganalisis aplikasi tanpa server Anda secara lokal sebelum mengunggahnya ke runtime Lambda. Untuk informasi selengkapnya, lihat Menguji dan men-debug aplikasi nirserver.	Selasa, 11 Agustus 2017
Wilayah Kanada (Pusat)	AWS Lambda sekarang tersedia di Wilayah Kanada (Tengah). Untuk informasi selengkapnya tentang wilayah Lambda dan titik akhir, lihat Wil ayah dan titik akhir di. Referensi Umum AWS	Selasa, 22 Juni 2017
Wilayah Amerika Selatan (São Paulo)	AWS Lambda sekarang tersedia di Wilayah Amerika Selatan (São Paulo). Untuk informasi selengkapnya tentang wilayah Lambda dan titik akhir, lihat Wil ayah dan titik akhir di. Referensi Umum AWS	6 Juni 2017
AWS Lambda dukungan untuk AWS X-Ray.	Lambda memperkenalkan dukungan untuk X-Ray, yang memungkinkan Anda mendeteksi, menganalisis, dan mengoptimalkan masalah kinerja dengan aplikasi Lambda Anda. Untuk informasi selengkapnya, lihat Visualisasikan pemanggilan Fungsi Acara Lambda menggunakan AWS X-Ray.	Selasa, 19 April 2017

Perubahan	Deskripsi	Date
Wilayah Asia Pasifik (Mumbai)	AWS Lambda sekarang tersedia di Wilayah Asia Pasifik (Mumbai). Untuk informasi selengkapnya tentang wilayah Lambda dan titik akhir, lihat Wil ayah dan titik akhir di. Referensi Umum AWS	28 Maret 2017
AWS Lambda sekarang mendukung Node.js runtime v6.10	AWS Lambda menambahkan dukungan untuk Node.js runtime v6.10. Untuk informasi selengkapnya, lihat Membangun Fungsi Acara Lambda dengan Node.js.	Selasa, 22 Maret 2017
Wilayah Eropa (London)	AWS Lambda sekarang tersedia di Wilayah Eropa (London). Untuk informasi selengkapnya tentang wilayah Lambda dan titik akhir, lihat Wil ayah dan titik akhir di. Referensi Umum AWS	1 Februari 2017
AWS Lambda dukungan untuk runtime .NET, Lambda @Edge (Pratinjau), Dead Letter Queues dan penyebaran otomatis aplikasi tanpa server.	AWS Lambda menambahkan dukungan untuk C #. Untuk informasi selengkapnya, lihat Membangun Fungsi Acara Lambda dengan C #. Lambda @Edge memungkinkan Anda menjalankan fungsi Lambda di lokasi AWS Edge sebagai respons terhadap CloudFront peristiwa. Untuk informasi selengkapnya, lihat Menyesuaikan di tepi dengan Lambda @Edge.	3 Desember 2016
AWS Lambda menambahkan Amazon Lex sebagai sumber acara yang didukung.	Dengan menggunakan Lambda dan Amazon Lex, Anda dapat dengan cepat membangun bot obrolan untuk berbagai layanan seperti Slack dan Facebook.	30 November 2016
Wilayah AS Barat (California Utara)	AWS Lambda sekarang tersedia di Wilayah AS Barat (California Utara). Untuk informasi selengkapnya tentang wilayah Lambda dan titik akhir, lihat Wil ayah dan titik akhir di. Referensi Umum AWS	21 November 2016

Perubahan	Deskripsi	Date
Memperkenalkan AWS SAM untuk membuat dan menyebarkan Lambda-based aplikasi dan menggunakan variabel lingkungan untuk pengaturan konfigurasi fungsi Lambda.	AWS SAM: Anda sekarang dapat menggunakan AWS SAM untuk menentukan sintaks untuk mengekspresikan sumber daya dalam aplikasi tanpa server. Untuk menerapkan aplikasi Anda, cukup tentukan sumber daya yang Anda butuhkan sebagai bagian dari aplikasi Anda, bersama dengan kebijakan izin terkait dalam file AWS CloudFormation template (ditulis dalam JSON atau YAML), kemas artefak penerapan Anda, dan terapkan template. Variabel lingkungan: Anda dapat menggunakan variabel lingkungan untuk menentukan pengaturan konfigurasi untuk fungsi Lambda Anda di luar kode fungsi Anda. Untuk informasi selengkapnya, lihat Bekerja dengan variabel lingkungan Lambda.	Selasa, 18 Nopember 2016
Wilayah Asia Pasifik (Seoul)	AWS Lambda sekarang tersedia di Wilayah Asia Pasifik (Seoul). Untuk informasi selengkapnya tentang wilayah Lambda dan titik akhir, lihat Wilayah dan titik akhir di. Referensi Umum AWS	29 Agustus 2016
Wilayah Asia Pacific (Sydney)	Lambda kini tersedia di Wilayah Asia Pacific (Sydney). Untuk informasi selengkapnya tentang wilayah Lambda dan titik akhir, lihat Wilayah dan titik akhir di. Referensi Umum AWS	Selasa, 23 Juni 2016
Pembaruan untuk konsol Lambda	Konsol Lambda telah diperbarui untuk menyederhanakan proses pembuatan peran.	23 Juni 2016
AWS Lambda sekarang mendukung Node.js runtime v4.3	AWS Lambda menambahkan dukungan untuk Node.js runtime v4.3. Untuk informasi selengkapnya, lihat Membangun Fungsi Acara Lambda dengan Node.js.	Selasa, 07 April 2016
Wilayah Eropa (Frankfurt)	Lambda kini tersedia di Wilayah Europe (Frankfurt). Untuk informasi selengkapnya tentang wilayah Lambda dan titik akhir, lihat Wilayah dan titik akhir di. Referensi Umum AWS	Selasa, 14 Maret 2016

Perubahan	Deskripsi	Date
Dukungan VPC	Anda sekarang dapat mengonfigurasi fungsi Lambda Anda untuk mengakses sumber daya di dalam VPC Anda. Untuk informasi selengkapnya, lihat Memberikan Lambda Event Functions akses ke sumber daya di Amazon VPC.	11 Februari 2016
Runtime Lambda sudah diperbarui.	Lingkungan eksekusi sudah diperbarui.	4 November 2015
Dukungan versioning, Python untuk mengembangkan kode untuk fungsi Lambda, peristiwa terjadwal, dan peningkatan waktu eksekusi	Sekarang Anda dapat mengembangkan kode fungsi Lambda menggunakan Python. Untuk informasi selengkapnya, lihat Membangun Fungsi Acara Lambda dengan Python. Versioning: Anda dapat mempertahankan satu atau beberapa versi fungsi Lambda Anda. Versioning versi memungkinkan Anda mengontrol versi fungsi Lambda mana yang dijalankan di lingkungan yang berbeda (misalnya, pengembangan, pengujian, atau produksi). Untuk informasi selengkapnya, lihat Kelola versi Fungsi Acara Lambda. Peristiwa terjadwal: Anda juga dapat menyiapkan Lambda untuk mengambil kode Anda secara rutin dan terjadwal menggunakan konsol Lambda. Anda dapat menentukan nilai tetap (jumlah jam, hari, atau minggu) atau Anda dapat menentukan persamaan cron. Untuk informasi selengkapnya, lihat Memanggil Fungsi Acara Lambda sesuai jadwal. Peningkatan dalam waktu eksekusi: Sekarang Anda dapat menyiapkan fungsi Lambda Anda untuk berjalan hingga lima menit yang memungkinkan fungsi berjalan lebih lama seperti data volume besar yang masuk dan pekerjaan pemrosesan.	08 Oktober 2015

Perubahan	Deskripsi	Date
Dukungan untuk DynamoDB Streams	DynamoDB Streams kini tersedia secara umum dan Anda dapat menggunakannya di semua wilayah tempat DynamoDB tersedia. Anda dapat mengaktifkan DynamoDB Streams untuk tabel Anda dan menggunakan fungsi Lambda sebagai pemicu untuk tabel. Pemicu adalah tindakan khusus yang Anda ambil dalam menanggapi pembaruan yang dibuat pada tabel DynamoDB. Untuk panduan contoh, lihat Tutorial: Menggunakan AWS Lambda dengan aliran Amazon DynamoDB.	14 Juli 2015
Lambda sekarang mendukung pemanggilan fungsi Lambda dengan klien. REST-compatible	Sampai sekarang, untuk memanggil fungsi Lambda Anda dari aplikasi web, seluler, atau IoT Anda, Anda memerlukan AWS SDK (misalnya, AWS SDK untuk Java, SDK untuk Android, atau AWS AWS SDK untuk iOS). Sekarang, Lambda mendukung pemanggilan fungsi Lambda dengan REST-compatible klien melalui API khusus yang dapat Anda buat menggunakan Amazon API Gateway. Anda dapat mengirim permintaan ke URL titik akhir fungsi Lambda Anda. Anda dapat mengonfigurasi keamanan pada titik akhir untuk mengizinkan akses terbuka, menggunakan AWS Identity and Access Management (IAM) untuk mengotorisasi akses, atau menggunakan kunci API untuk mengukur akses ke fungsi Lambda Anda oleh orang lain. Untuk contoh latihan Memulai, lihat Memanggil Fungsi Peristiwa Lambda menggunakan titik akhir Amazon API Gateway.	09 Juli 2015
Konsol Lambda kini menyediakan cetak biru untuk membuat fungsi Lambda dan mengujinya dengan mudah.	Konsol Lambda menyediakan serangkaian cetak biru. Setiap cetak biru menyediakan contoh konfigurasi sumber peristiwa dan kode sampel untuk fungsi Lambda Anda yang dapat Anda gunakan untuk membuat Lambda-based aplikasi dengan mudah. Semua latihan Lambda Memulai sekarang menggunakan cetak biru.	09 Juli 2015

Perubahan	Deskripsi	Date
Lambda sekarang mendukung Java untuk membuat fungsi Lambda Anda.	Sekarang Anda dapat menulis kode Lambda di Java. Untuk informasi selengkapnya, lihat Membangun Fungsi Acara Lambda dengan Java.	15 Juni 2015
Lambda kini mendukung penentuan objek Amazon S3 sebagai fungsi .zip saat membuat atau memperbarui fungsi Lambda.	Anda dapat mengunggah paket deployment fungsi Lambda (file .zip) ke bucket Amazon S3 di wilayah yang sama tempat Anda ingin membuat fungsi Lambda. Kemudian, Anda dapat menentukan nama bucket dan nama kunci objek saat Anda membuat atau memperbarui fungsi Lambda.	28 Mei 2015
Lambda sekarang tersedia secara umum dengan dukungan tambahan untuk backend seluler	Lambda saat ini secara umum tersedia untuk penggunaan produksi. Rilis ini juga memperkenalkan fitur baru yang mempermudah pembuatan backend seluler, tablet, dan Internet of Things (IoT) menggunakan Lambda yang menskalakan secara otomatis tanpa menyediakan atau mengelola infrastruktur. Lambda sekarang mendukung peristiwa waktu nyata (sinkron) dan peristiwa asinkron. Fitur tambahan termasuk konfigurasi dan manajemen sumber peristiwa yang lebih mudah. Model izin dan model pemrograman telah disederhanakan dengan pengenalan kebijakan sumber daya untuk fungsi Lambda Anda.	Selasa, 09 April 2015
Rilis pratinjau	Rilis pratinjau dari Panduan Developer AWS Lambda .	13 November 2014

Terjemahan disediakan oleh mesin penerjemah. Jika konten terjemahan yang diberikan bertentangan dengan versi bahasa Inggris aslinya, utamakan versi bahasa Inggris.